Lời Nói Đầu

Tài liệu này được thực hiện sau một thời gian làm việc về Tính toán hiệu năng cao của nhóm nghiên cứu High Performance Computing tại Trung tâm Tính Toán Hiệu Năng Cao trường Đại Học Bách Khoa Hà Nội.

Trong tài liệu này, chúng tôi xin trình bày, đặc tả chi tiết về kiến trúc cũng như các chức năng của **BKluster** – một hệ thống tính toán song song ghép cụm dựa trên kiến trúc hệ Beowulf và mô hình lập trình truyền thông điệp cùng với bộ phần mềm **BKlusware** - một tập các phần mềm hỗ trợ tối đa nhiều người dùng sử dụng hệ thống BKluster ở nhiều mức độ khác nhau. Nội dung của tài liệu được chia thành 9 chương:

Chương 1: Trình bày tổng quan về máy tính song song mô hình ghép cụm, giới thiệu về mục đích và các hướng tiếp cận của nhóm nghiên cứu khi xây dựng hệ thống BKluster và gói phần mềm BKlusware.

Các chương 2, 3, 4: trình bày về Hệ thống phát triển tích hợp bao gồm Bộ công cụ soạn thảo và quản lý mã nguồn, Bộ công cụ gỡ rối chương trình song song và Các nghiên cứu, thử nghiệm xây dựng máy ảo cùng ngôn ngữ thông dịch đơn giản giúp phát triển các ứng dụng song song.

Chương 5 mô tả thiết kế và hoạt động của Hệ thống thực thi tính toán. Chức năng của hệ thống này là đệ trình, theo dõi và quản lý kết quả trả về của các ứng dụng song song

Bốn chương cuối: chương 6, chương 7, chương 8, chương 9 mô tả chi tiết Hệ thống quản trị và giám sát. Hệ thống này có các chức năng: Quản lý các gói phần mềm, Quản lý cấu hình cluster, Quản lý người dùng, Giám sát hoạt động và Đánh giá hiệu năng hệ thống.

Mục Lục

Danh mục hình

Danh mục bảng

CHƯƠNG 1 Hệ Thống Tính Toán Song Song Ghép Cụm BKluster và Gói Phần Mềm BKlusware

1.1 Mô hình song song ghép cụm (cluster)

Ngày nay, những nghiên cứu, thực nghiệm về mô phỏng, dự báo đã dẫn đến việc cần phải giải quyết những bài toán có số lượng phép tính rất lớn trong những khoảng thời gian chấp nhận được. Một trong những giải pháp cho vấn đề trên là áp dụng kỹ thuật tính toán song song phát triển những chương trình chạy trên các siêu máy tính. Theo Top500, website xếp hạng các siêu máy tính mạnh nhất Thế Giới, siêu máy tính có thể được chia thành 3 loại chính: MPP (Massively Parallel Processors), Cluster (Hệ thống tính toán song song ghép cụm) và Constellation (cluster với các nút tính toán là các máy tính nhiều vi xử lý).

** Một số định nghĩa:*

- **Máy tính song song** là một tập các tài nguyên tính toán có khả năng truyền thông và kết hợp với nhau để giải quyết các bài toán lớn trong khoảng thời gian chấp nhận được (Almasi và Gottlieb – 1989).
- **Cluster** (hệ thống tính toán song song ghép cụm) là một hệ thống tính toán phân tán bao gồm nhiều máy tính được ghép nối với nhau qua mạng và được sử dụng như một tài nguyên tính toán duy nhất (*G.F.Pfister. In Search of Cluster (second edition). Prentice Hall NJ.1998).*

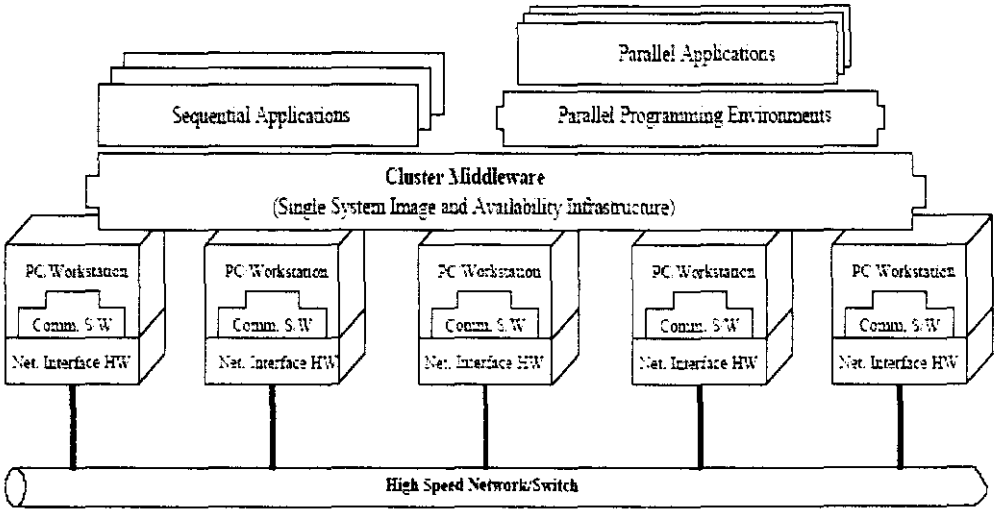
Qua định nghĩa trên, ta có thể thấy, hệ thống tính toán song song ghép cụm là một máy tính song song, trong đó:

- Các tài nguyên tính toán là bộ vi xử lý và bộ nhớ trong tại mỗi máy tính,
- Các tài nguyên tính toán này có khả năng truyền thông và kết hợp với nhau thông qua cáp mạng. Thường thì quy mô của hệ thống chỉ giới hạn trong một mạng cục bộ (LAN), trong đó có một máy tính đóng vai trò máy chủ (server), các máy tính còn lại đóng vai trò nút tính toán (computing node)

Việc thiết lập hệ thống tính toán song song ghép cụm từ những máy tính có cấu trúc đơn giản, sử dụng các công nghệ mạng phổ biến đã được bắt đầu từ năm 1994 với công trình nghiên cứu của Thomas Sterling cùng Donal Becker. Với 16 máy tính PC, vi xử lý DX và card mạng Ethernet 10 Mbps, Sterling và Becker đã xây dựng thành công hệ thống tính toán song song ghép cụm đầu tiên trên Thế Giới – Hệ Beowulf. Ngày nay, Beowulf đã trở thành một trong những chuẩn phổ biến nhất

trên Thế Giới để xây dựng cluster. So với kiến trúc MPP, kiến trúc cluster có ưu điểm là dễ mở rộng quy mô, thích hợp với nhiều công nghệ chế tạo vi xử lý, chế tạo bộ nhớ, công nghệ mạng phổ biến. So với kiến trúc Constellation, kiến trúc cluster có ưu điểm là các nút tính toán có thể là những máy tính đơn giản, thông dụng. Tuy nhiên, kiến trúc cluster cũng có những hạn chế như: quá trình triển khai, cấu hình hệ thống tương đối phức tạp, hệ thống hoạt động không ổn định, và nhược điểm lớn nhất là vấn đề truyền thông giữa các nút tính toán. Việc nghiên cứu nhằm nâng cao khả năng truyền thông là một trong những vấn đề quan trọng hàng đầu trong quá trình phát triển các cấu trúc, mô hình hệ thống phân cụm. Các hệ thống song song ghép cụm hiện đại đều dùng các kiến trúc mạng tiên tiến như: Myrinet, Gigabit Ethernet,... nhằm nâng cao tốc độ truyền thông giữa các nút tính toán.

Trên thực tế, cơ sở hạ tầng phần cứng mới chỉ là một phần trong toàn thể mô hình chung của tính toán phân cụm. Cùng với sự phát triển của lĩnh vực Công nghệ thông tin trên thế giới, phần mềm đang ngày càng chứng minh được vai trò cốt yếu của mình trong các hệ thống song song ghép cụm. Đối với các hệ thống tính toán ghép cụm, phần mềm càng có vai trò quan trọng hơn. Chính các phần mềm đảm bảo cho hệ thống gồm nhiều máy tính riêng lẻ có thể hoạt động ổn định và cộng tác hiệu quả. Một hệ thống tính toán song song ghép cụm (dựa trên mạng LAN) hoạt động như một hệ thống đơn dưới cái nhìn của người dùng và ứng dụng.



Hình 2-1 Kiến trúc mô hình tính toán ghép cụm

Kiến trúc song song ghép cụm cho phép thiết lập nên những máy tính song song từ những máy tính có kiến trúc đơn giản, công nghệ mạng phổ biến. Điều này rất thích

hợp với những nước đang phát triển, các trường Đại Học. Thực tế cho thấy, từ những máy tính PC và công nghệ mạng Fast Ethernet hoặc Gigabit Ethernet, có thể xây dựng nên những hệ thống tính toán ghép cụm có hiệu năng cực đại RMAX vào cỡ hàng chục Gigaflop. Mô hình phổ biến nhất hiện nay để xây dựng hệ thống tính toán song song ghép cụm từ những máy tính có kiến trúc đơn giản, công nghệ mạng phổ biến chính là mô hình Beowulf.

Beowulf là một loại cluster với những yêu cầu sau (phân biệt Beowulf với các loại cluster khác):

- Các nút tính toán trong cluster chỉ phục vụ cho mục đích của cluster, không làm một chức năng nào khác.
- Mạng liên kết các nút tính toán chỉ phục vụ cho cluster, không làm thêm một chức năng nào khác.
- Các nút tính toán có chi phí rẻ, phổ biến trên thị trường, dễ nâng cấp hoặc thay thế (*M2COTS: Mass Market Commodity-Off-The-Self)*.
- Hệ thống mạng liên kết phổ biến, tương thích với nhiều kiến trúc máy tính.
- Mọi phần mềm cài đặt trên các nút đều là phần mềm nguồn mở.
- Mục đích của toàn bộ hệ thống là tính toán hiệu năng cao (HPC).

Một số đặc điểm phổ biến của hệ Beowulf

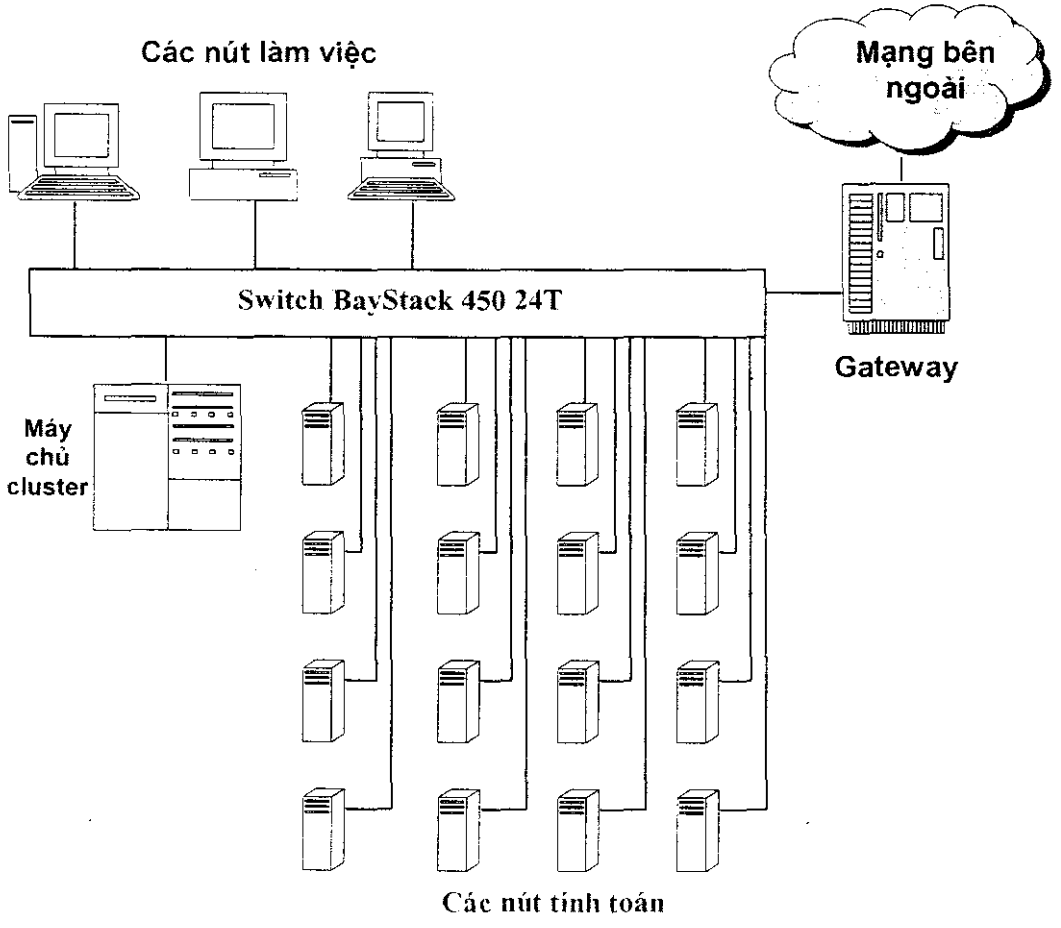
- Các nút đều sử dụng hệ điều hành Linux. Đặc điểm này đã có tính lịch sử lâu đời với hệ Beowulf đầu tiên xây dựng dựa trên Linux và GNU với các ethernet driver do Donnal Becker viết cho Linux.
- Trong hệ thống có một máy được gọi là máy chủ. Ở mức độ vừa đủ, mọi tương tác với toàn bộ hệ thống đều được thực hiện thông qua máy chủ (do đó trong toàn bộ hệ thống, chỉ duy nhất máy chủ là cần bàn phím và màn hình). Máy chủ sẽ có hai đường kết nối: kết nối với môi trường bên ngoài (đến các hệ thống khác hoặc Internet) và kết nối với các nút tính toán trong cùng hệ thống. Máy chủ còn hay được dùng để chia sẻ bộ nhớ ngoài với các nút tính toán (nfs).
- Thông thường cấu hình của các nút tính toán trong cùng một hệ tương đối giống nhau (tuy nhiên điều này không phải là quá cần thiết).
- Thông thường, tại một thời điểm xác định, một nút chỉ thực hiện một nhiệm vụ tính toán.

1.2 Hệ thống tính toán song song ghép cụm BKluster

BKluster là một hệ thống tính toán song song ghép cụm xây dựng theo mô hình hệ Beowulf từ những máy tính PC hoặc HP NetServer, công nghệ mạng Fast Ethernet. BKluster được thiết kế với những mục đích chính sau:

- Cho phép người sử dụng soạn thảo, quản lý và biên dịch mã nguồn của các ứng dụng song song theo chuẩn MPI.
- Cho phép người sử dụng đệ trình chương trình song song dưới dạng các công việc (các job), theo dõi và quản lý kết quả trả về của quá trình thực hiện job.
- Cung cấp cho người quản trị hệ thống các công cụ trực quan để quản lý cấu hình, quản lý người sử dụng và đánh giá hiệu năng của hệ thống.

Toàn bộ hệ thống được kết nối theo kiến trúc mạng như sau:



Hình 2-2 Kiến trúc mạng ghép nối của hệ thống BKluster

1.2.1 Kết nối các máy tính trong BKluster

Về mặt phần cứng, các máy tính trong hệ thống BKluster được kết nối với nhau theo topo mạng Fast Ethernet, tốc độ truyền tối đa là 100Mbps. Hệ điều hành cài đặt trên máy tính là Linux Fedora Core I. Trước khi cài đặt và triển khai các gói phần mềm phục vụ tính toán song song, cần phải cấu hình và thiết lập một số dịch vụ sau trong toàn bộ mạng LAN:

- **Đặt địa chỉ IP cho các máy tính**: các máy tính trong mạng được đặt địa chỉ IP lớp C, danh sách địa chỉ IP và tên máy tương ứng được đặt trong file /etc/hosts.
- **Kích hoạt một số dịch vụ cho phép truy nhập và chạy ứng dụng từ xa**: Một số ứng dụng chạy trên hệ điều hành Linux như rsh, rlogin, ssh, kerberos, ... cho phép người sử dụng truy cập và chạy ứng dụng trên 1 máy tính khác từ một máy tính bất kỳ. Trong BKluster, các khả năng kích hoạt và chạy các ứng dụng từ xa được sử dụng nhằm hai mục đích:
  - Cung cấp khả năng sử dụng, điều khiển các nút tính toán thông qua 1 máy chủ duy nhất (với mô hình Beowulf, chỉ duy nhất máy chủ là cần bàn phím, màn hình,...).
  - Cho phép các tiến trình của cùng một chương trình song song có thể trao đổi thao tinh với nhau cũng như thu thập kết quả trả về tiến trình chủ.

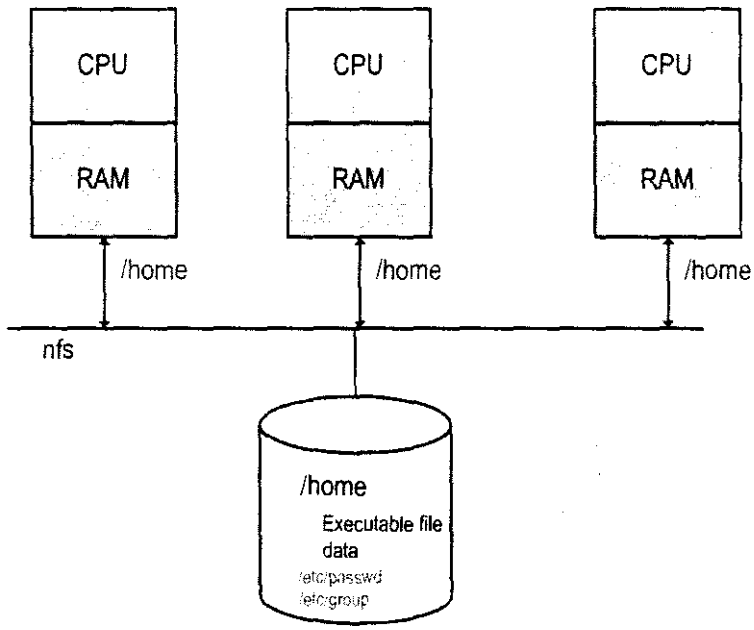
  Hệ thống BKluster sử dụng hai dịch vụ là rsh và rlogin. Cần lưu ý một điểm, để có thể chạy các ứng dụng song song với MPI, cần phải bỏ cơ chế xác thực người sử dụng giữa các máy tính bằng cách liệt kê danh sách tên các máy tính trong cùng một mạng LAN trong file /etc/hosts.equiv.
- **Thiết lập hệ thống file mạng NFS**: NFS được xây dựng dựa trên việc gọi thủ tục từ xa, cho phép sử dụng các tệp tin trên các trạm ở xa giống như chúng ở trên trạm cục bộ. NFS bao gồm chương trình client trên máy cần sử dụng các tệp tin và chương trình server cung cấp các tệp tin. NFS có thể được sử dụng trên nhiều hệ thống khác nhau (Redhat, Debian,...) việc truy cập các tệp tin là trong suốt đối với người sử dụng. NFS cho phép dữ liệu dành cho người sử dụng có thể đặt trên một trạm tập trung, các trạm khác có thể thiết lập thư mục này lúc khởi động (thông qua lệnh mount).

  NFS cho phép dữ liệu chiếm nhiều dung lượng đĩa có thể được giữ tại một trạm và phân phối cho các trạm khác. Tuy nhiên, tốc độ truy cập bộ nhớ ngoài sẽ phụ thuộc vào tốc độ đường truyền và cấu hình của mạng LAN.

Trong hệ thống BKluster, NFS được sử dụng để thiết lập thư mục home của những người sử dụng hệ thống. Đây là một vùng nhớ ngoài dùng chung đặt trên máy chủ cluster, giúp các nút tính toán trong hệ thống có thể truy cập, nạp chương trình chạy vào bộ nhớ trong, đọc dữ liệu để xử lý ...

- **Thiết lập hệ thống thông tin mạng NIS**: NIS sung cấp các tiện ích truy cập cơ sở dữ liệu để phân phối thông tin, chẳng hạn như dữ liệu trong /etc/passwds và /etc/groups cho tất cả các trạm trong mạng. Điều này làm cho mạng trở nên như một hệ thống duy nhất.

Sau khi thiết lập xong các dịch vụ trên, toàn bộ hệ thống có thể được coi như một máy tính song song với các tài nguyên tính toán là một tập các vi xử lý & bộ nhớ trong phân tán kết nối với nhau bằng cáp mạng LAN, toàn bộ hệ thống dùng chung một vùng nhớ ngoài duy nhất và một hệ thống người quản trị duy nhất.



Hình 2-3 BKluster như một máy tính song song nhiều vi xử lý

1.2.2 Thiết lập môi trường tính toán song song

Một chương trình song song viết theo mô hình truyền thông điệp (Message Passing) bao gồm nhiều tiến trình chạy trên các nút tính toán, trao đổi thông tin với nhau trong quá trình hoạt động. Sau khi thực hiện các bước cấu hình mạng LAN, thiết lập các dịch vụ ở trên, toàn bộ hệ thống đã có thể coi như một máy tính song song duy nhất, các vi xử lý đã có các khả năng sau:

- Nhận biết các vi xử lý khác
- Truy xuất vào cùng một vùng nhớ ngoài

Tuy nhiên, để có thể biên dịch và thực thi các chương trình song song, hệ thống vẫn còn thiếu ba chức năng sau:

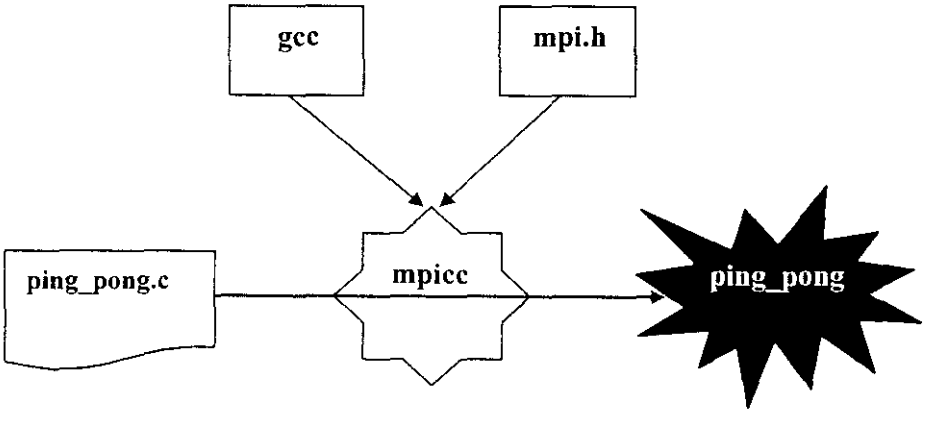
- Biên dịch các chương trình song song
- Chạy chương trình song song một tập các nút tính toán
- Theo dõi quá trình chạy của các tiến trình của chương trình song song, thực hiện việc truyền thông giữa các tiến trình, quản lý kết quả trả về.

Cùng với sự phát triển của mô hình song song ghép cụm, đã có rất nhiều phần mềm hay gói phần mềm được xây dựng để thực hiện các chức năng trên (pvm, các phiên bản MPI,...). Hệ thống BKluster sử dụng gói phần mềm LAM/MPI để thực hiện việc thiết lập môi trường tính toán, biên dịch và chạy các ứng dụng song song. Sau khi cài đặt và cấu hình, LAM/MPI cung cấp cho người sử dụng các module sau:

- Các script hỗ trợ việc biên dịch chương trình viết bằng các ngôn ngữ bậc cao như: C, C++, Fortran theo chuẩn MPI.
- Các tiến trình ngầm (daemon) đảm nhiệm việc quản lý và truyền thông điệp giữa các tiến trình song song. Các thông điệp được truyền thông qua giao thức TCP/IP

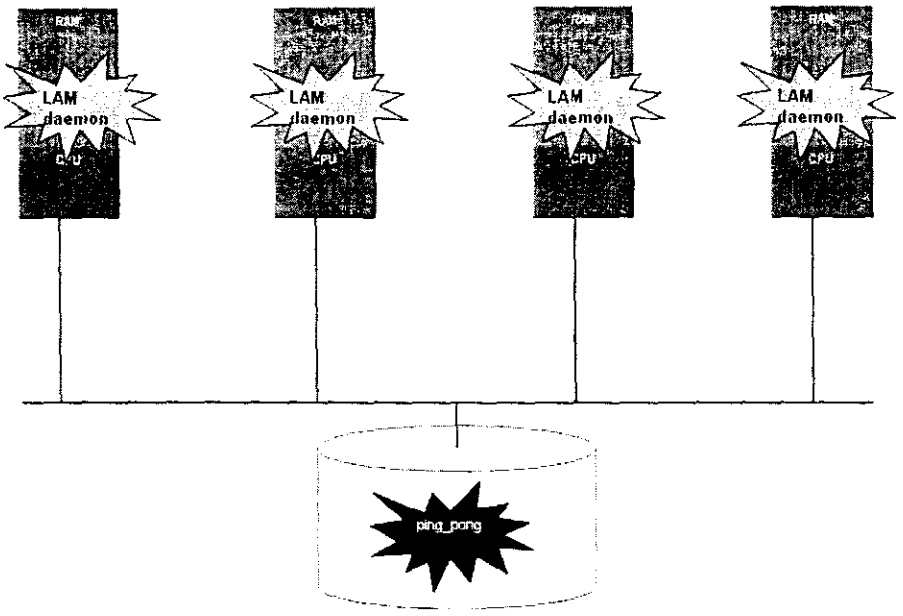
Giả sử có một chương trình viết theo chuẩn MPI tên là ping_pong.c. Dưới đây là quá trình biên dịch và chạy ping_pong.c với LAM/MPI:

Bước 1: Biên dịch ping_pong.c thành file chạy (các file executable đối với hệ điều hành Linux):



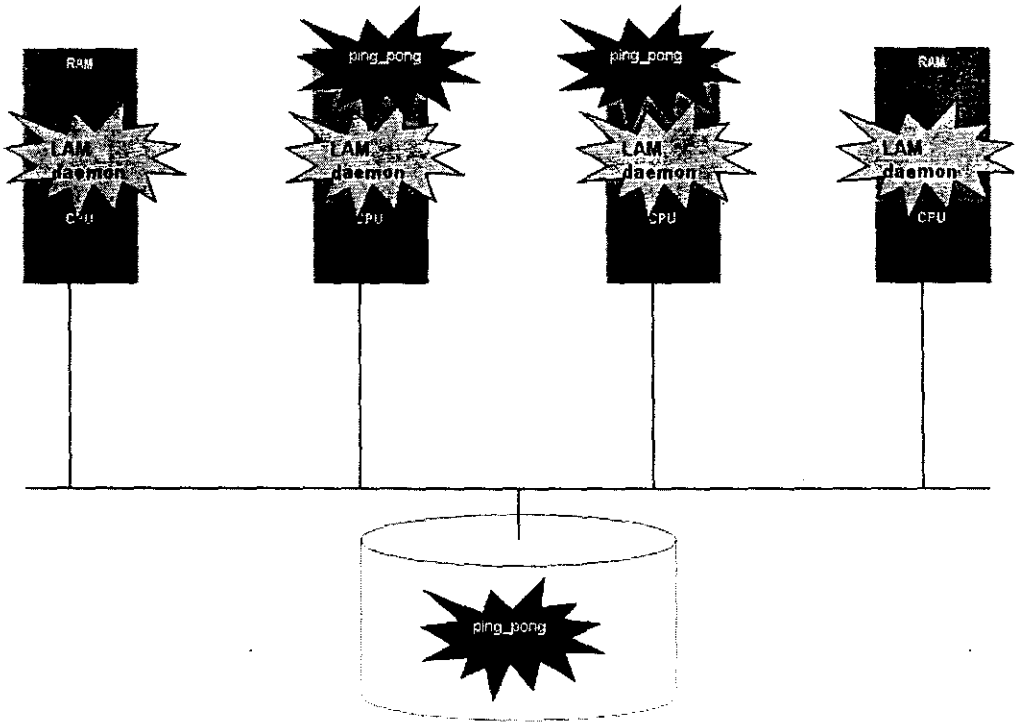
Hình 2-4 Biên dịch chương trình song song với script mpicc

Bước 2: Khởi động môi trường tính toán song song bằng cách chạy các LAM daemon dưới dạng các tiến trình ngầm (lệnh lamboot). Các LAM daemon sẽ có chức năng chạy chương trình và truyền thông điệp giữa các tiến trình. Lưu ý rằng, bước 1 và bước 2 hoàn toàn độc lập với nhau.



Hình 2-5 Khởi động môi trường song song với các LAM daemon

Bước 3: Chạy chương trình ping_pong với một số nút tính toán xác định (lệnh mpirun). Các vi xử lý sẽ tiến hành đọc và nạp nội dung ping_pong vào bộ nhớ trong của nút tính toán (hình vẽ minh họa chạy chương trình với hai nút tính toán). Mỗi tiến trình sẽ được đánh một số thứ tự gọi là rank.



Hình 2-6 Các vi xử lý đọc và nạp chương trình song song vào bộ nhớ trong tương ứng

Bước 4: Các tiến trình sẽ thực hiện các lệnh tương ứng với rank của mình. Quá trình đóng gói và truyền thông điệp, điều khiển các tiến trình sẽ được thực hiện thông qua các LAM daemon.

Với gói phần mềm LAM/MPI, hệ thống BKluster đã hoàn toàn có thể chạy được những chương trình song song trên các nút tính toán của mình. Tuy nhiên, người sử dụng hoàn toàn phải thao tác qua dòng lệnh, đồng thời phải biết rõ về số lượng, địa chỉ IP cũng như tên các nút tính toán trong hệ thống.

1.2.3 Cài đặt hệ thống quản lý tài nguyên và phân tải

Một vấn đề quan trọng đặt ra với các hệ thống đóng vai trò dịch vụ tính toán đó là phân bổ các tiến trình tính toán trên tài nguyên của hệ thống sao cho hiệu quả hoạt động của toàn bộ hệ thống là tối ưu. Để làm được việc này, trong hệ thống cần cài đặt các phần mềm có chức năng quản lý tài nguyên và lập lịch. Hai loại phần mềm này thường được tích hợp chung vào một gói phần mềm gọi là "**hệ quản lý tài nguyên và phân tải**". Trong hệ thống BKluster, hệ quản lý tài nguyên và phân tải được sử dụng là PBS – Portable Batch System. PBS là phần mềm mã nguồn mở của NASA. Hiện nay, PBS đang được ứng dụng và phát triển trong nhiều hệ thống tính toán song song ghép cụm trên Thế Giới.

PBS là một gói phần mềm mã nguồn mở được cung cấp kèm theo đầy đủ tài liệu đặc tả, hướng dẫn sử dụng. Những đặc điểm trên giúp cho việc cài đặt và cấu hình PBS có thể tiến hành một cách tương đối dễ dàng, đồng thời cũng tạo điều kiện cho việc cải tiến, tích hợp thêm các chức năng chuyên biệt cho phù hợp với mục đích sử dụng cụ thể.

Hệ thống quản lý tài nguyên và phân tải PBS bao gồm 3 module chính :

+ **pbs_server**: nhận và điều phối các yêu cầu tính toán.
+ **pbs_scheduler**: lập lịch cho các yêu cầu tính toán.
+ **pbs_mom**: quản lý tài nguyên và thực thi công việc trên các nút tính toán. Việc quản lý tài nguyên và thực thi công việc là hai quá trình độc lập với nhau.

Khi tiến hành cài đặt, thông thường hai môđun pbs_server và pbs_scheduler được cài đặt trên máy chủ, pbs_mom được cài đặt trên các nút tính toán. Cả 3 môđun trên đều hoạt động dưới dạng các tiến trình ngầm – daemon.

Quá trình hoạt động của toàn bộ hệ thống như sau:

1. Người sử dụng đệ trình 1 công việc lên pbs_server.
2. pbs_server yêu cầu pbs_scheduler lập lịch.

3. pbs_scheduler yêu cầu pbs_server và pbs_mom các thông tin về trạng thái và tài nguyên.

4. pbs_server và pbs_mom chuyển các thông tin về trạng thái và tài nguyên cho pbs_scheduler.

5. pbs_scheduler tiến hành lập lịch và chuyển kết quả về cho pbs_server.

   Kết quả lập lịch có thể là 1 trong 2 trường hợp sau :

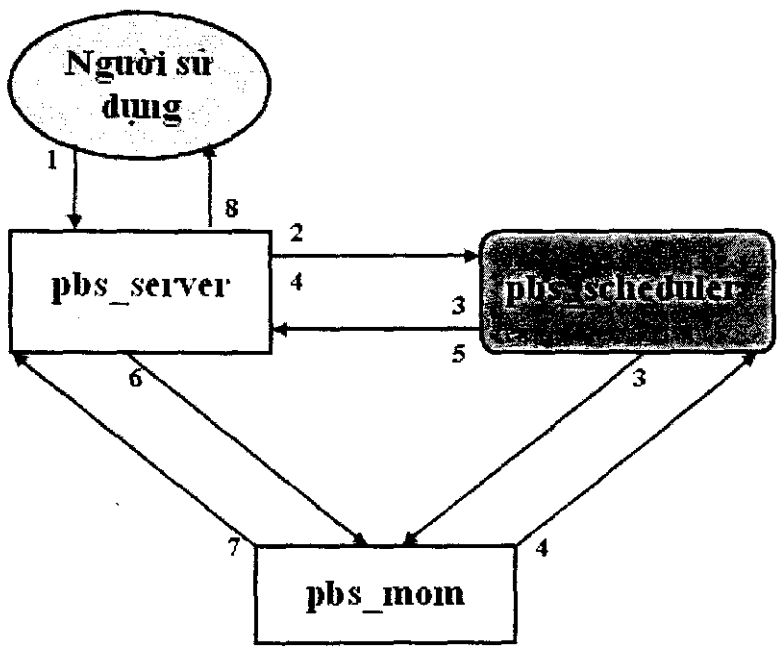
   + Xác định rõ tên công việc được chạy và danh sách các nút tính toán tương ứng

   + Không có công việc nào được chạy, trong trường hợp này công việc mới sẽ được đưa và hàng đợi.

6. Trong trường hợp có công việc được chạy, pbs_server sẽ phát tín hiệu yêu cầu các pbs_mom thực hiện công việc trên các nút tính toán.

7. Khi công việc kết thúc, pbs_mom chuyển kết quả cho pbs_server.

8. pbs_server trả kết quả về cho người sử dụng.



Hình 2-7 Hoạt động của hệ thống quản lý tài nguyên và phân tải PBS

Trong gói phần mềm PBS còn có các thư viện lập trình và một tập các câu lệnh cho phép người quản trị và người sử dụng dễ dàng tương tác với hệ thống ở chế độ dòng lệnh hoặc qua các chương trình viết bằng ngôn ngữ bậc cao. Với những đặc

điểm trên, PBS đã đáp ứng được những yêu cầu cơ bản của người quản trị và người sử dụng hệ thống tính toán song song ghép cụm.

1.2.4 Cài đặt hệ thống giám sát hoạt động

Ở mức độ cơ bản, gói phần mềm PBS đã đáp ứng được đầy đủ yêu cầu của việc quản lý tài nguyên cũng như phân tải trong Bkluster. Tuy nhiên, PBS không hỗ trợ mạnh giao diện đồ họa, các thông tin về tài nguyên hệ thống được lưu trữ dưới dạng các struct, hỗ trợ ngôn ngữ C. Do đó, trong hệ thống BKluster, PBS chỉ được sử dụng khả năng lập lịch và phân tải, còn chức năng quản lý tài nguyên, giám sát hoạt động của hệ thống được thực hiện bởi gói phần mềm Ganglia kết hợp với cơ sở dữ liệu Round Robin.

Ganglia gồm 2 thành phần chính

- ***Gmond*** : chạy tại các nút tính toán là một tiến trình ngầm có nhiệm vụ thu thập các thông tin về trạng thái tài nguyên tại các nút tính toán này rồi gửi các thông tin này dưới dạng XML ra một cổng nào đó để thành phần khác có thể đọc và sử dụng các thông tin này.

- ***Gmetad*** : Đây là một tiến trình ngầm của Ganglia. Thành phần này chạy tại nút chủ của hệ thống có nhiệm vụ thu thập thông tin của tất cả các gmond tại các nút tính toán. Do đó có thể nói thông tin của gmetad là thông tin về toàn bộ hệ thống. Thông tin này được gửi cho thành phần server phía trên và được lưu trong cơ sơ dữ liệu Round Robin.

RRDTool (Round Robin Database Tool): Các thông tin về tài nguyên các nút trong hệ thống được lưu trữ dưới dạng cơ sở dữ liệu Round Robin. Phần mềm quản lý cơ sở dữ liệu này là RRDTool. Đây là cơ sở dữ liệu lưu các giá trị của một hoặc nhiều đại lượng biến đổi theo thời gian, ví dụ như thông lượng mạng trung bình sau mỗi 5 giây, phần trăm sử dụng bộ vi xử lý của máy tính, dung lượng bộ nhớ còn trống... Cơ sở dữ liệu này gồm hai thành phần : đó là các phần tử dữ liệu và một con trỏ trỏ tới phần tử hiện hành. Có thể hình dung như một hình tròn có các điểm phân bố đều trên đường biên của nó, các điểm này chính là nơi mà dữ liệu được lưu trữ. Kẻ một bán kính từ tâm tới các điểm này ta sẽ có con trỏ. Mỗi khi các dữ liệu được đọc hoặc ghi, con trỏ sẽ di chuyển đến phần tử tiếp theo. Và bởi vì chúng ta đang trên một đường tròn nên không có điểm khởi đầu cũng không có điểm kết thúc. Sau một khoảng thời gian tất cả các phần tử trên biên của hình tròn đều được sử dụng hết, các vị trí cũ sẽ được tái sử dụng một các tự động. Chính vì thế nên

kích thước của cơ sở dữ liệu không bị phình ra theo thời gian.

Một tiện ích khác của RRDTool là khả năng tạo ra các biểu đồ từ các giá trị lưu trong cơ sở dữ liệu Round Robin. Do đó chúng ta có thể dễ dàng theo dõi sự biến đổi của một hay nhiều đại lượng theo thời gian một cách trực quan nhất bằng hình ảnh.

1.3 Gói phần mềm BKlusware

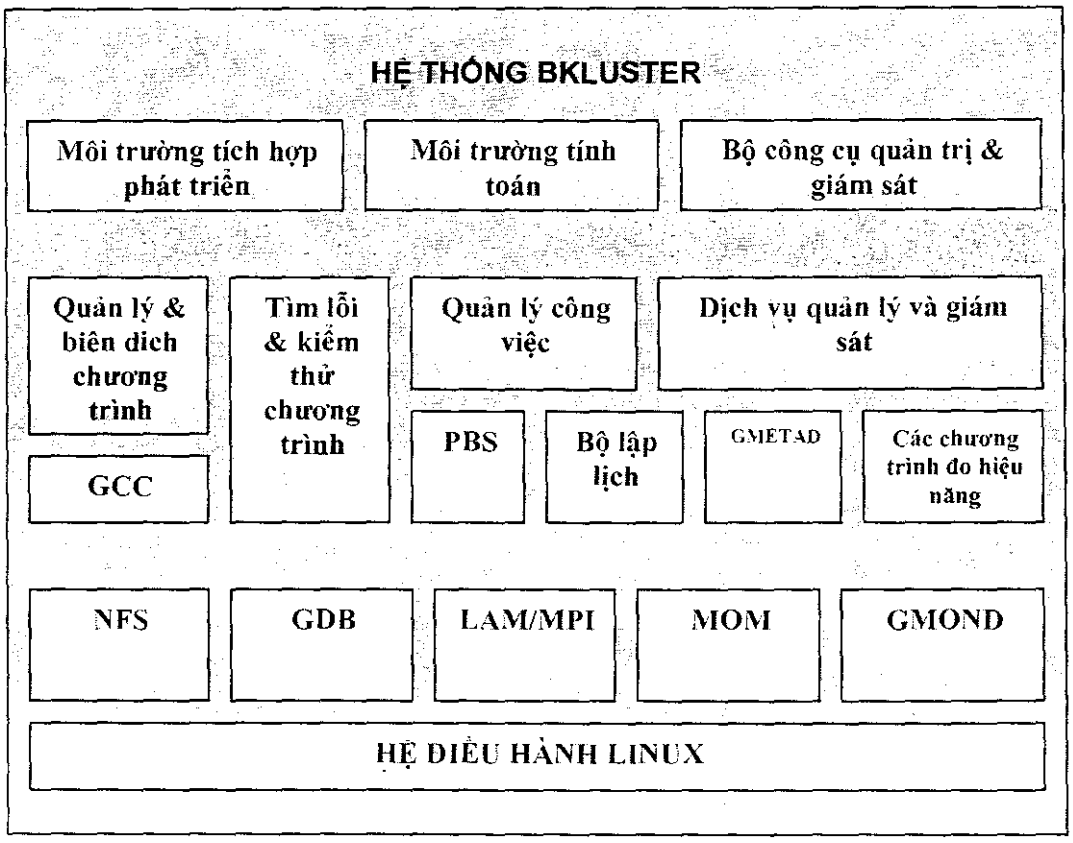
Với việc cấu hình như trên, sau khi cài đặt các gói phần mềm LAM/MPI, PBS, Ganglia, hệ thống đã đáp ứng được các yêu cầu cơ bản của việc phát triển, biên dịch, lập lịch chạy các ứng dụng song song, cũng như cung cấp cho người quản trị các công cụ để quản lý, cấu hình, theo dõi toàn bộ hệ thống. Tuy nhiên, việc sử dụng, vận hành những gói phần mềm trên còn đòi hỏi người sử dụng phải có những kiến thức nhất định về hệ thống, cũng như phải biết khá nhiều các tùy chọn khác nhau tương ứng với từng câu lệnh. Mặt khác, các gói phần mềm trên là do những nhà cung cấp khác nhau xây dựng và phát triển nên chỉ hỗ trợ nhau rất hạn chế hoặc không hỗ trợ.

Xuất phát từ các quan sát đó, Trung tâm Tính Toán Hiệu Năng Cao trường Đại Học Bách Khoa Hà Nội đã tiến hành xây dựng gói phần mềm BKlusware với các mục đích sau:

- Liên kết các gói phần mềm sẵn có trên Thế Giới hoạt động một cách thống nhất, cung cấp các chức năng phát triển tích hợp, thực thi công việc và quản lý giám sát hệ thống tính toán song song ghép cụm.
- Hỗ trợ người sử dụng nhiều mức: BKluster hỗ trợ sử dụng nhiều mức, người dùng thuần tuý, người phát triển chuyên nghiệp và người quản trị hệ thống. Người dùng thuần tuý khai thác hệ thống dựa trên các phần mềm tính toán có sẵn hoặc đưa ra yêu cầu tính toán dựa trên một ngôn ngữ đặc tả của hệ thống đề ra. Đối với nhóm người dùng này, khái niệm song song là trong suốt. Hệ thống cũng cho phép người dùng phát triển các phần mềm tự viết các chương trình tính toán nhằm tối ưu hoá dựa trên đặc trưng của từng bài toán cụ thể. Các chương trình được phát triển theo mô hình lập trình truyền thông điệp. Tuy nhiên, ở mức độ này người dùng cũng có thể bỏ qua một số

vấn đề phức tạp trong việc tính toán song song như việc phân tải, sắp đặt các chương trình song song trên các nút tính toán. Nói cách khác, sự phức tạp trong tổ chức và các thành phần hệ thống đã được che lấp giúp người dùng nhìn BKluster như một máy tính nhiều bộ vi xử lý. Lớp người dùng cuối cùng – người quản trị mạng buộc phải nhìn hệ thống một cách chân thực – một tập các máy tính được "liên kết" dựa trên môi trường truyền thông. Các công cụ quản trị và theo được xây dựng không phải để che đậy sự phức tạp của hệ thống. Nó chỉ giúp người dùng giảm bớt khối lượng công việc phải làm của người quản trị cũng như hiện thị một cách trực quan về hệ thống, giúp người quản trị có những quyết định đúng đắn.

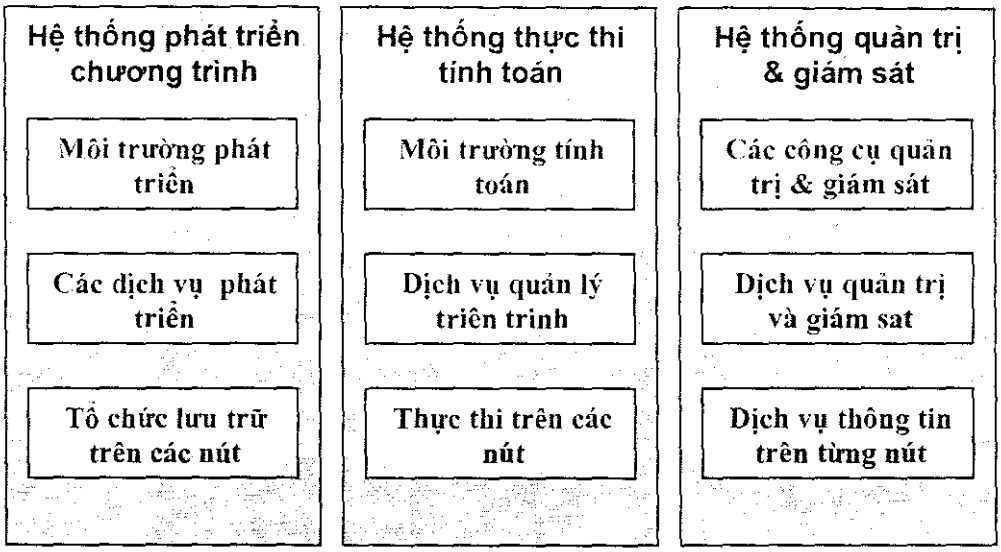
Tập hợp các phần mềm giúp BKluster trở thành một hệ thống tính toán gọi là BKlusware.



Hình 2-8 Kiến trúc phân tầng hệ thống BKluster

Về mặt chức năng chính, gói phần mềm BKlusware có 3 chức năng chính, tương ứng với 3 hệ thống nhỏ hơn:

- Hệ thống Phát triển tích hợp
- Hệ thống Thực thi tính toán
- Hệ thống Giám sát và quản trị



Hình 2-9 Kiến trúc phân hệ BKluster

Nội dung các chương tiếp theo sẽ đi vào trình bày cụ thể ba hệ thống trên:

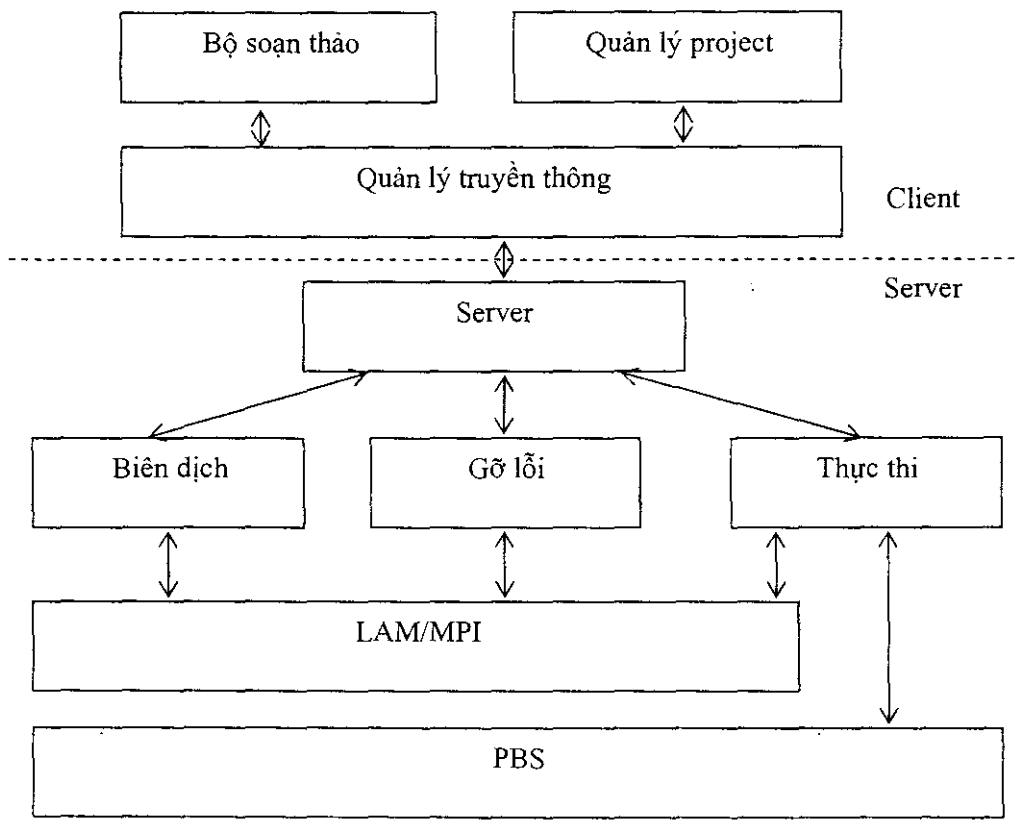
- Hệ thống phát triển tích hợp bao gồm Bộ công cụ soạn thảo và quản lý mã nguồn, Bộ công cụ gỡ rối chương trình song song và Các nghiên cứu, thử nghiệm xây dựng máy ảo cùng ngôn ngữ thông dịch đơn giản giúp phát triển các ứng dụng song song.
- Hệ thống thực thi tính toán cho phép đệ trình, theo dõi và quản lý kết quả trả về của các ứng dụng song song
- Hệ thống quản trị và giám sát cung cấp các chức năng: Quản lý các gói phần mềm, Quản lý cấu hình cluster, Quản lý người dùng, Giám sát hoạt động và Đánh giá hiệu năng hệ thống

CHƯƠNG 2 Bộ Công Cụ Soạn Thảo và Quản Lý Mã Nguồn

2.1 Đặt vấn đề

Tính toán song song không phải là một lĩnh vực mới, nhưng hiện nay ở nước ta việc áp dụng tính toán song song còn rất hạn chế một phần do các điều kiện về vật chất và một phần do việc tiếp cận và áp dụng công nghệ. Các hệ thống tính toán song song thường có kiến trúc phức tạp khiến cho việc mô hình hoá và lập trình các bài toán đòi hỏi tính chuyên nghiệp và sự hiểu biết sâu về tính toán song song. Do vậy việc xây dựng một môi trường phát triển ứng dụng song song là rất cần thiết tạo cơ sở cho việc ứng dụng tính toán song song trong khoa học kỹ thuật và trong cuộc sống.

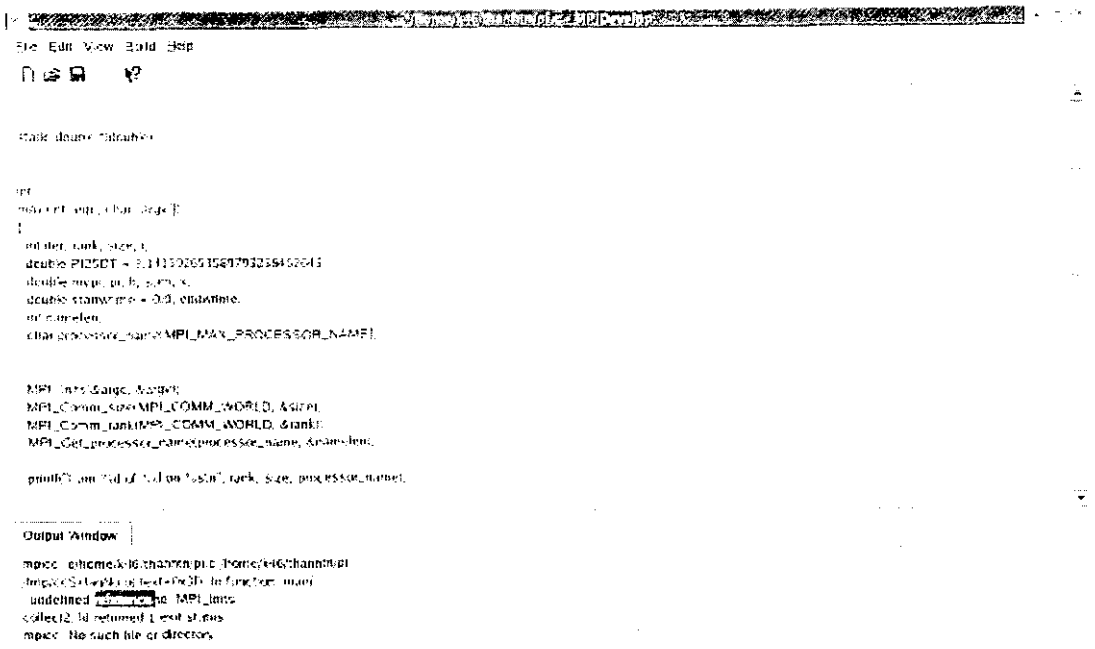
2.2 Kiến trúc môi trường phát triển các ứng dụng song song



Hình 2-1 Kiến trúc mô hình MPIDevelop

Môi trường phát triển ứng dụng song song MPIDevelop là một môi trường tích hợp được xây dựa trên mô hình Client/Server. Người lập trình sẽ soạn thảo mã nguồn trên một bộ soạn thảo nằm ở máy client. Mã nguồn sẽ được ghi vào các tệp trên server, dịch, liên kết thư viện và chạy trên server. Server có một bộ tiếp nhận yêu cầu, phân loại và chuyển các yêu cầu này đến các bộ phận khác để thực hiện. Mô hình này đảm bảo đáp ứng được nhiều người lập trình trong cùng một thời điểm.

- **Bộ soạn thảo** là thành phần nằm trên client giúp người lập trình soạn thảo các lệnh cho ứng dụng.
- **Quản lý project:** thực hiện việc quản lý các file mã nguồn, các lớp, hàm trong mã nguồn.
- **Quản lý truyền thông:** thực hiện truyền nhận dữ liệu giữa client và server thông qua socket.
- **Server** là một tiến trình ngầm chạy trên một nút trong hệ thống tính toán song song tiếp nhận nhận mọi yêu cầu gửi đến từ client, phân loại rồi chuyển cho các thành phần khác thực hiện. Mỗi người sử dụng có một tài khoản dùng để đăng nhập vào hệ thống. Tài khoản này tương ứng với một thư mục được gọi là thư mục home của người sử dụng. Người sử dụng chỉ được phép thao tác trên các tệp hoặc thư mục nằm trong thư mục home của người đó. Khi một người sửu dụng đăng nhập vào hệ thống, bộ tiếp nhận yêu cầu sẽ đọc các tham số cần thiết của người sử dụng trong thư mục home và tạo ra một phiên làm việc (session) độc lập.
- **Bộ biên dịch:** thực hiện việc biên dịch mã nguồn mà lập trình viên đã soạn thảo. Hiện tại chương trình chỉ cho phép biên dịch các chương trình song song được viết theo chuẩn MPI.
- **Bộ thực thi** cho phép người lập trình chạy thử chương trình vừa được tạo ra bằng cách cho chạy file thực thi được tạo ra khi build. Thường thì các chương trình chạy phải có tham số, tham số này có thể nhập từ dòng lệnh hoặc môi trường phát triển tích hợp có thêm mục cho phép lập trình viên nhập trước tham số khi chạy chương trình.



Hình 2-2 Giao diện chương trình MPIDevelop

2.3 Kết quả đạt được

MPIDevelop hiện tại là phiên bản thử nghiệm cho phép lập trình viên soạn thảo mã nguồn, biên dịch và thực thi chương trình. Các chức năng này đã được hoàn thành một cách cơ bản, tuy nhiên việc biên dịch thực thi vẫn chưa được tách ra để thực hiện trên Server. Có thể sử dụng chương trình để lập trình một số bài toán tính toán song song chạy trên hệ thống BKCluster.

2.4 Kết luận

MPIDevelop hiện giờ mới chỉ đạt được một số chức năng cơ bản và đang trong quá trình hoàn thiện tiếp. Các công việc cần thực hiện để hoàn thiện MPIDevelop:

- Hoàn chỉnh bộ soạn thảo mã nguồn với các chức năng nâng cao hỗ trợ lập trình viên soạn thảo.
- Thiết kế bộ quản lý project cho phép lập trình viên xem các file mã nguồn, các lớp, các hàm trong chương trình.

- Thiết kế phần server thực hiện việc biên dịch, gỡ lỗi, thực thi chương trình từ xa.
- Tích hợp thêm bộ trợ giúp cho người lập trình.

CHƯƠNG 3 Xây Dựng Bộ Công Cụ Gỡ Rối Chương Trình Song Song

Công việc phát triển ứng dụng – hay lập trình – là công việc luôn luôn có lỗi (bug). Việc xác định các lỗi tiêu tốn rất nhiều thời gian của người lập trình. Do đó nhu cầu về một công cụ hỗ trợ người lập trình có thể nhanh chóng phát hiện ra các lỗi mà họ mắc phải là thực sự cần thiết. Đối với lập trình song song, vai trò của công cụ gỡ lỗi càng quan trọng hơn nữa do khả năng xảy ra lỗi trong chương trình song song cao hơn nhiều so với trong chương trình tuần tự. Hơn thế, các lỗi xảy ra trong lập trình song song cũng phức tạp hơn so với trong chương trình tuần tự. Có sự trợ giúp của phần mềm hỗ trợ gỡ rối, công việc của người lập trình sẽ trở nên dễ dàng hơn rất nhiều.

3.1 Gỡ rối cho các chương trình song song

3.1.1 Sơ lược chung về gỡ rối

3.1.1.1 Vấn đề gỡ rối trong lập trình

Trong phát triển phần mềm, một nhiệm vụ cơ bản nhưng không hề dễ dàng là đảm bảo sự chính xác, đúng đắn của chương trình. Việc xuất hiện lỗi trong chương trình là khó tránh khỏi do bản chất tự nhiên của con người là không hoàn hảo. Do đó, khi con người vẫn còn tham dự vào công việc lập trình, lỗi sẽ vẫn còn xuất hiện.

Có nhiều kỹ thuật để tìm và sửa lỗi trong chương trình. Một kỹ thuật cơ bản và đơn giản hay được các lập trình viên sử dụng là dùng câu lệnh in ra màn hình. Câu lệnh được đặt tại vị trí nghi ngờ có lỗi, và in ra màn hình giá trị các biến theo yêu cầu nhằm giúp cho lập trình viên có thể xác định lỗi của chương trình. Tuy nhiên phương pháp này chỉ áp dụng cho các chương trình bé và với những lỗi đơn giản. Khi gặp chương trình có quy mô lớn, phương pháp trở nên khó khăn hơn rất nhiều vì việc tìm vị trí nghi ngờ có lỗi cũng như đặt các câu lệnh in ra màn hình đòi hỏi nhiều công sức của lập trình viên. Ngoài ra một điểm yếu rất rõ của phương pháp này là tính bị động: người lập trình không có khả năng tương tác với chương trình, không thể thay đổi giá trị các biến để kiểm tra hoạt động của chương trình.

Một phương pháp thứ hai để gỡ lỗi chương trình là sử dụng các công cụ hỗ trợ gỡ lỗi. Trong những năm gần đây, rất nhiều công cụ phần mềm đã được tạo ra nhằm trợ giúp lập trình viên trong việc này (VD: dbx, gdb, sdb, DEBUG, ...). Các chương trình gỡ lỗi này thông thường cho phép lập trình viên dừng chương trình khi đang chạy, kiểm tra trạng thái của chương trình, cho phép sửa đổi giá trị các biến cũng như giá trị ngăn xếp, in ra các giá trị có thể liên quan đến lỗi, và tiếp tục chạy chương trình. Việc cho phép điều khiển chương trình chạy một cách tương tác

như vậy giúp cho các lập trình viên có thể tìm ra lỗi một cách hiệu quả.

3.1.1.2 Công cụ gdb

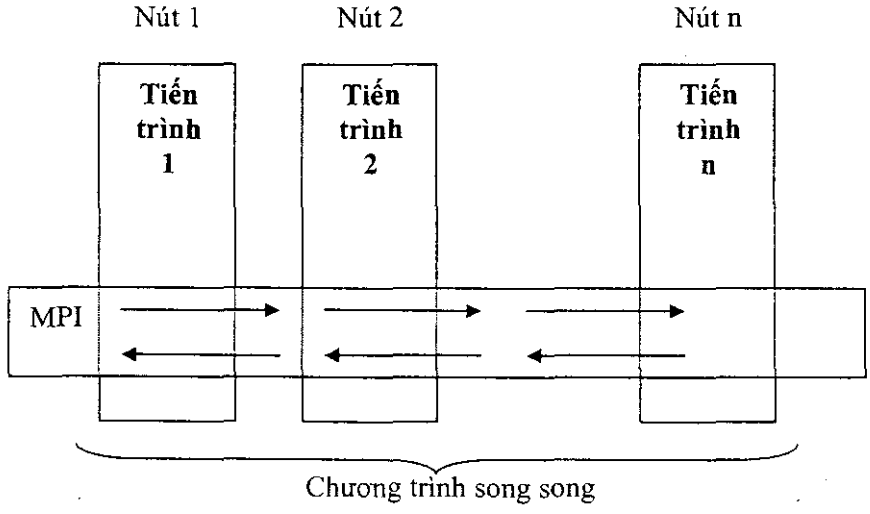
Một công cụ hỗ trợ gỡ lỗi rất thông dụng và được dùng phổ biến trong thế gới Linux là gdb. Gdb là một chương trình phần mềm mã nguồn mở giao diện dòng lệnh – CLI (command line interface) cho phép người sử dụng điều khiển qua trình chạy của chương trình cần gỡ lỗi một cách tương tác. Gdb có khả năng thực hiện bốn công việc chính sau (cộng thêm các khả năng khác hỗ trợ cho bốn khả năng chính này):

- Chạy chương trình, thay đổi các thông số ban đầu để tác động lên hoạt động của chương trình.
- Dừng chương trình tại những điều kiện cụ thể.
- Kiểm tra trạng thái chương trình khi dừng lại.
- Thay đổi các giá trị trong chương trình để chẩn đoán lỗi.

3.1.2 Những bài toán được đặt ra trong gỡ rối chương trình song song

3.1.2.1 Một số đặc điểm của chương trình song song

Như chúng ta đã biết, một chương trình song song bao gồm nhiều tiến trình chạy trên các nút tính toán, trao đổi thông tin với nhau trong quá trình hoạt động, với mục đích đưa ra kết quả cuối cùng với hiệu năng cao hơn nhiều lần so với chương trình tuần tự.



Hình 3-1 Mô hình hoạt động của chương trình song song

3.1.2.2 Các vấn đề trong gỡ rối chương trình song song

Khi chuyển từ bài toán gỡ rối các chương trình tuần tự sang bài toán gỡ rối các chương trình song song, ta gặp nhiều vấn đề mới nảy sinh. Các vấn đề này xuất phát từ các đặc điểm đặc trưng của các chương trình song song:

- Chương trình song song không phải chỉ bao gồm một luồng công việc được thực hiện một cách có thứ tự như trong chương trình tuần tự. Chúng bao gồm nhiều luồng chương trình song song chạy đồng thời trong các ràng buộc về thời gian và dữ liệu.
- Trong quá trình hoạt động, các luồng chương trình song song này quan hệ với nhau rất chặt chẽ thông qua việc trao đổi thông điệp.
- Khái niệm trạng thái của chương trình song song có nhiều khác biệt so với chương trình tuần tự. Tại một thời điểm, trạng thái của chương trình song song bao gồm một tập các trạng thái của các tiến trình trên các nút tính toán. Tập các trạng thái này thay đổi đối với mỗi lần chạy chương trình, tạo ra tính hỗn loạn, gây khó khăn cho người lập trình khi bắt trạng thái để gỡ lỗi.

Do vậy, việc thiết kế, tổ chức và viết một chương trình song song là không hề đơn giản. Một chút sai sót rất nhỏ (chẳng hạn như quên không đặt điểm đồng bộ giữa các tiến trình, bị lỗi deadlock - một lỗi rất hay gặp trong lập trình song song) có thể gây ra những lỗi ngầm bên trong rất khó phát hiện. Vì vậy, công việc lập trình song song dễ phát sinh lỗi hơn trong lập trình tuần tự, và các lỗi cũng thường phức tạp hơn.

Phương pháp gỡ lỗi rất phổ biến trong lập trình tuần tự là sử dụng câu lệnh in ra màn hình (chẳng hạn printf() trong ngôn ngữ C). Phương pháp này rất đơn giản và tỏ ra khá hiệu quả khi gỡ lỗi chương trình tuần tự. Tuy nhiên, đối với chương trình song song, phương pháp gỡ lỗi sử dụng câu lệnh in ra màn hình lại không hề hiệu quả, thậm chí nhiều khi còn khiến cho người gỡ lỗi chương trình bị lạc hướng. Điều này xuất phát từ đặc điểm của chương trình song song là bao gồm nhiều tiến trình cùng chạy song song. Khi thực hiện đến câu lệnh in ra màn hình, các tiến trình song song này sẽ "tranh nhau hiển thị", tiến trình nào đến trước thì hiển thị trước, tiến trình nào đến sau thì hiển thị sau. Do đó, thứ tự thực hiện câu lệnh in ra màn hình là rất lộn xộn. Dựa vào những gì hiện ra trên màn hình khi này để gỡ lỗi chương trình song song là rất khó khăn. Ngoài ra phương pháp này cũng rất khó có thể phát hiện ra các lỗi xuất phát từ vấn đề truyền thông giữa các tiến trình, chẳng hạn như vấn đề deadlock.

Phương pháp gỡ lỗi chuyên nghiệp hơn là sử dụng các phần mềm hỗ trợ gỡ lỗi chương trình như gdb, dbx, sdb, DEBUG, Turbo system debugger. Tuy nhiên, tất cả các chương trình kể trên đều là các chương trình gỡ lỗi dành cho chương trình

tuần tự. Cần phải có những giải pháp phù hợp để có thể áp dụng các công cụ phần mềm này vào gỡ rối các chương trình song song.

Một vấn đề nữa cần chú ý đối với chương trình song song là việc trao đổi thông tin giữa các tiến trình. Đây là điểm đặc trưng khác biệt của các chương trình song song. Quá trình trao đổi thông tin này cũng gây ra nhiều lỗi. Do đó, một công cụ cho phép theo dõi quá trình truyền thông giữa các tiến trình đóng một vai trò quan trọng trong phát triển các công cụ gỡ lỗi song song.

Hiện nay trên thế giới có rất ít các chương trình gỡ lỗi hoàn chỉnh dành cho chương trình song song. Một chương trình hoàn thiện nhất mà ta có thể kể ra là Total View. Đây là công cụ hiệu quả trong việc gỡ lỗi chương trình song song, và nó đã được LAM/MPI chính thức hỗ trợ. Nhưng một điều đáng tiếc Total View là một chương trình thương mại. Nó không phải là chương trình mã nguồn mở như LAM, do đó không phù hợp với BKluster và không có nhiều ý nghĩa trong việc học tập nghiên cứu.

3.1.3 Một số hướng tiếp cận

Hiện nay trên thế giới có hai hướng tiếp cận chính đối với việc gỡ rối chương trình song song: theo mô hình gỡ rối tuần tự truyền thống và theo hướng mô hình truyền thông điệp.

Đối với mô hình theo hướng gỡ rối tuần tự truyền thống, công việc gỡ rối được tiến hành theo hai pha:

- Pha 1: tiến hành gỡ rối đối với từng tiến trình trên từng nút tính toán giống như đối với gỡ rối chương trình tuần tự.
- Pha 2: tổng hợp kết quả từ các nút tính toán để đưa ra cái nhìn tổng quát về hoạt động của cả chương trình cho người sử dụng.

Trong hai pha của mô hình gỡ rối này, pha thứ hai chính là thành phần quan trọng tạo nên sự khác nhau giữa gỡ rối tuần tự và gỡ rối song song. Đối với các bài toán song song theo mô hình SIMD, khi các tiến trình cùng thực hiện một chuỗi các lệnh như nhau thì công việc tổng hợp có thể đơn giản hơn. Hệ thống sẽ giống như một máy đơn thao tác cùng một lúc trên rất nhiều dữ liệu. Công việc sẽ trở nên phức tạp hơn đối với bài toán song song theo mô hình MIMD. Do khi này các tiến trình thực hiện các công việc khác nhau trên các bộ dữ liệu khác nhau nên công việc tổng hợp đòi hỏi nhiều công sức. Thậm chí, hệ thống có thể còn yêu cầu tri thức để có thể đưa ra những kết quả tổng hợp đúng đắn và chính xác.

Đối với hướng mô hình truyền thông điệp, công việc gỡ rối chú trọng vào theo dõi quá trình truyền thông giữa các tiến trình. Xuất phát từ quan điểm các lỗi phổ biến và khó phát hiện trong các chương trình song song thường là lỗi của quá trình

truyền thông, những người phát triển công cụ gỡ rối theo hướng này đã xây dựng nên những công cụ cho phép theo dõi hiệu quả quá trình truyền thông giữa các tiến trình, bao gồm các chức năng chính:

- Theo dõi bản đồ truyền thông điệp: thời điểm truyền thông, nơi xuất phát và đích đến của các thông điệp.
- Kiểm tra nội dung của các thông điệp

Sử dụng công cụ gỡ rối theo hướng này, người lập trình sẽ dễ dàng hơn trong việc phát hiện ra những lỗi đặc trưng của chương trình song song.

Trong hai hướng tiếp cận trên, hướng tiếp cận thứ nhất tạo cho người sử dụng khả năng chủ động trong công việc gỡ rối chương trình. Người sử dụng có thể chạy chương trình, dừng chương trình, theo dõi và thay đổi giá trị các biến để kiểm tra phản ứng của chương trình. Việc chẩn đoán lỗi sẽ trở nên đơn giản hơn.

Nhược điểm của hướng tiếp cận thứ nhất là sự phức tạp khi xây dựng công cụ. Như đã được phát biểu ở trên, việc tổng hợp kết quả từ các tiến trình đơn lẻ để đưa ra cái nhìn chung tổng quát, trực quan của cả chương trình là không đơn giản, cần nhiều công sức nghiên cứu.

Đối với hướng tiếp cận thứ hai, công cụ hỗ trợ theo dõi truyền thông điệp cho phép người sử dụng quan sát một cách trực quan quá trình hoạt động của các tiến trình song song, việc truyền thông điệp giữa chúng và nội dung của các thông điệp. Những khả năng này giúp cho người lập trình dễ dàng hơn trong việc phát hiện các lỗi truyền thông.

Tuy nhiên, nhược điểm của các công cụ này là không cho phép người sử dụng tương tác với chương trình. Người dùng chỉ được phép theo dõi quá trình truyền thông chứ không được can thiệp vào quá trình đó.

Phần mềm hỗ trợ gỡ rối BKPD sẽ sử dụng cả hai hướng tiếp cận trên. Một mặt, BKPD sử dụng gdb để tiến hành gỡ rối đơn lẻ trên các tiến trình, sau đó tổng hợp lại và cung cấp cái nhìn tổng quan cho người sử dụng. Mặt khác, BKPD cũng bao gồm thành phần BKViewer để cung cấp cho người dùng cái nhìn trực quan về quá trình truyền thông điệp trong chương trình song song.

3.2 BKPD – HỆ THỐNG GỠ RỐI CHO CÁC CHƯƠNG TRÌNH SONG SONG TRUYỀN THÔNG ĐIỆP

3.2.1 Hệ thống tính toán song song BKluster

Hệ thống BKluster được xây dựng dựa trên nền môi trường lập trình song song truyền thông điệp LAM/MPI - theo chuẩn MPI, và bộ quản lý và phân tải PBS.

MPI - Giao diện truyền thông điệp - Message Passing Interface (MPI)) – là một chuẩn quy định các hàm giao diện lập trình ứng dụng (API), cho phép người lập trình viết các chương trình song song theo mô hình truyền thông điệp.

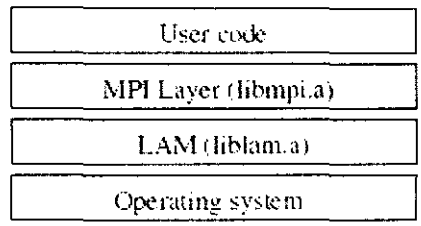
MPI là kết quả của nhiều thập kỷ nghiên cứu trong lĩnh vực tính toán song song, được đề xuất bởi nhóm phần mềm mã nguồn mở MPI Forum.

MPI có thể thực hiện trên rất nhiều môi trường khác nhau, từ các môi trường lớn như các máy tính song song (IBM SP, SGI Origin) đến các môi trường nhỏ hơn như các Cluster. Đặc biệt chuẩn MPI được sử dụng nhiều trong các trường đại học, nơi chủ yếu áp dụng mô hình tính toán song song Cluster. Các chuẩn MPI được thiết kế hỗ trợ tính khả chuyển và không phụ thuộc platform.

LAM MPI là một gói phần mềm hệ thống và bộ thư viện lập trình mã nguồn mở, tuân theo các chuẩn MPI. LAM được phát triển bởi phòng thí nghiệm các hệ thống mở (Open Systems Lab) tại trường đại học Indiana.

LAM MPI không chỉ là thư viện thực hiện các ứng dụng MPI, mà nó còn tạo ra môi trường LAM (LAM run-time enviroment). Đây là môi trường ngầm mức hệ thống cung cấp rất nhiều dịch vụ cho các ứng dụng MPI.

Gói phần mềm LAM có hai thành phần chính: môi trường thực thi LAM và tầng truyền thông MPI. Nói chung tầng LAM cung cấp các dịch vụ xương sống cho tầng truyền thông MPI, do đó tầng MPI phụ thuộc vào tầng LAM. Hai thành phần này nằm trong hai thư viện riêng biệt là liblam.a và libmpi.a.

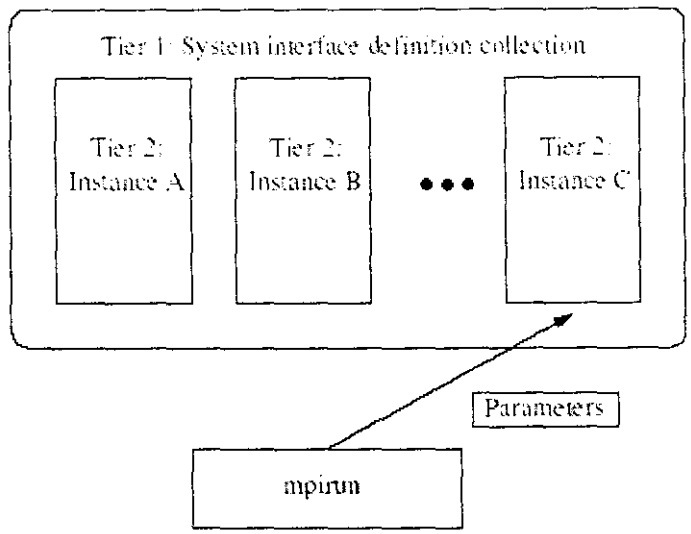


Hình 3-1. Các thành phần của LAM/MPI

Cả hai thành phần chính trên của gói LAM MPI đều được thiết kế theo mô hình kiến trúc thành phần – có khả năng mở rộng cao, khả năng chọn lựa và cấu hình các module nhỏ tại thời điểm thực thi (run-time). Kiến trúc thành phần này của

LAM MPI được gọi là giao diện dịch vụ hệ thống – System Service Interface (SSI).

LAM SSI được thiết kế theo mô hình module hai tầng (two tier). Mục tiêu của SSI là cho phép lựa chọn trong nhiều mẫu giao diện tại thời điểm thực thi chương trình và truyền các thông số cho mẫu đó. Hình dưới mô tả cấu trúc hai tầng này



Hình 3-2. Mô hình module hai tầng của LAM SSI

Kiến trúc này cung cấp khả năng "plug-n-play" cho các dịch vụ của LAM và MPI. Mỗi thực thể của tier 2 là một tập code riêng biệt, được cấu hình và cài đặt độc lập với các thực thể khác. LAM sẽ tự động tìm và cấu hình các thực thể được lựa chọn vào môi trường hoạt động chung của mình. Như vậy có thể thêm các thực thể dịch vụ mới vào LAM một cách dễ dàng.

Hiện nay LAM bao gồm bốn nhóm giao diện SSI, trong đó mỗi nhóm bao gồm nhiều thực thể module SSI cho phép chọn lựa khi thực thi LAM. Bốn nhóm này là:

Tên	Vị trí	Mô tả
Boot	LAM	Boot SSI sử dụng để khởi tạo các tiến trình môi trường tại các nút trong mạng, với các lệnh lamboot, recon, wipe, và lamgrow. Boot SSI dùng các môi trường thực thi khác (như ssh/rsh) để khởi tạo môi trường của chính mình.
Coll	MPI	Coll SSI cung cấp các giải thuật nền cho các hàm truyền thông quảng bá (collective communication) trong MPI.

Cr	LAM và MPI	Hỗ trợ checkpoint/restart. Cho phép tạm dừng và khôi phục chương trình song song.
Rpi	MPI	MPI Request Progression Interface (RPI) SSI sử dụng trong tất cả các truyền thống điểm-điểm trong MPI. Đây là tầng thấp nhất trong MPI phục vụ truyền thông, đảm nhận công việc truyền các byte giữa các tiến trình MPI.

Bảng 3-1 Các SSI của LAM/MPI

3.2.2 Những hỗ trợ của LAM/MPI trong gỡ lỗi các chương trình song song

LAM/MPI có hai đặc điểm hỗ trợ cho quá trình gỡ rối các chương trình MPI. Thứ nhất, LAM được thiết kế cho phép người lập trình song song có khả năng gỡ lỗi một cách tương tác, dựa trên công cụ gỡ lỗi tuần tự gdb. Việc gỡ lỗi được thực hiện bằng cách gắn gdb vào các tiến trình song song, sau đó tiến hành gỡ lỗi qua gdb, giống như đối với chương trình tuần tự.

Trong LAM, một chương trình song song được chạy khi triệu gọi câu lệnh mpirun như sau:

```
mpirun C myparallelprogram
```

Thứ tự các công việc thực hiện của LAM khi chạy chương trình song song là:

- Khởi tạo các biến môi trường cần thiết cho việc chạy chương trình song song. Nói chung các biến này đều chỉ phục vụ riêng cho LAM/MPI. Có một biến môi trường khá có ích là biến LAMRANK trên mỗi máy chứa định danh của máy đó trong môi trường LAM.
- Thực hiện chạy chương trình được chỉ ra trong câu lệnh mpirun trên tất cả các máy. Ở bước này thì chương trình tuần tự hay chương trình song song đều như nhau: chúng đều có n mẫu và được chạy cùng lúc trên trong n tiến trình trên m máy (do mỗi máy có thể có nhiều tiến trình)
- Đối với chương trình song song, đặc trưng song song chỉ bắt đầu được thể hiện khi các máy cùng chạy đến lệnh MPI_Init. Khi các máy chạy lệnh này, chúng sẽ thiết lập môi trường truyền thông, trao đổi thông tin cho nhau để nhận biết tất cả các máy trong môi trường và thu thập chỉ số của các máy. Sau khi thu thập xong các thông tin trên thì MPI_Init kết thúc. Như vậy có thể coi MPI_Init là một hàm truyền thông quảng bá, là một điểm đồng bộ

của chương trình song song. Các tiến trình trên các máy phải chờ nhau tại điểm này.

- Đối với các chương trình tuần tự, do không có MPI_Init nên chương trình sẽ chạy thẳng một mạch đến khi kết thúc. Khi đó do MPI thấy chương trình đã kết thúc mà vẫn chưa có MPI_Init nên nó sẽ báo lỗi.

Ta có thể ứng dụng nguyên tắc này để chạy gdb trên các tiến trình như sau:

```
mpirun C gdb myparallelprogram
```

Như vậy mpirun được dùng không phải để chạy chương trình song song mà để chạy gdb trên các nút tính toán. Dựa vào qui trình chạy chương trình song song trong môi trường Lam được nêu ở trên ta thấy điều này là hợp lệ. Khi đó gdb được chạy đồng thời trên tất cả các nút, và chương trình song song cần chạy thì được điều khiển bởi gdb. Người dùng có thể gỡ lỗi các tiến trình song song trên các nút thông qua các tiến trình gdb chạy trên các nút đó.

Tuy nhiên, để gỡ rối người lập trình phải tự mình thực hiện từng lệnh trên từng tiến trình riêng lẻ. Nó không cung cấp cho người sử dụng cái nhìn tổng quan về toàn bộ chương trình, và gây nhiều bất tiện trong quá trình gỡ rối.

Đặc điểm thứ hai LAM hỗ trợ cho quá trình gỡ rối là nó cho phép tạo ra file vết truyền thông giữa các tiến trình (trace file). Đây là file lưu lại các thông tin trao đổi giữa các tiến trình. Nhờ file này, ta có thể tái hiện lại quá trình truyền thông của chương trình MPI.

3.2.3 Phân tích yêu cầu của chương trình gỡ rối song song BKPD

BKPD sử dụng cả hai khả năng hỗ trợ của LAM/MPI trong gỡ rối các chương trình song song. Mục tiêu của chương trình hỗ trợ gỡ rỗi song song BKPD là tạo ra một công cụ phần mềm:

- Cho phép người sử dụng tiến hành gỡ lỗi các tiến trình song song MPI trên các nút tính toán một cách tập trung.
- Cho phép người sử dụng tiến hành gỡ lỗi thông qua một giao diện đồ hoạ - GUI (Graphic User Interface).
- Cho phép người sử dụng tiến hành gỡ lỗi một cách tương tác, có khả năng theo dõi, điều khiển quá trình hoạt động của chương trình. Người sử dụng trong quá trình gỡ rối có khả năng thay đổi các giá trị biến cũng như ngăn xếp để có thể chẩn đoán được lỗi.
- Cho phép người sử dụng tiến hành gỡ lỗi từ xa (phân tán) thông qua mạng Internet, qua một giao diện chạy trên Windows.

- Cho phép người sử dụng theo dõi khung cảnh truyền thông điệp giữa các tiến trình thông qua công cụ BKViewer.

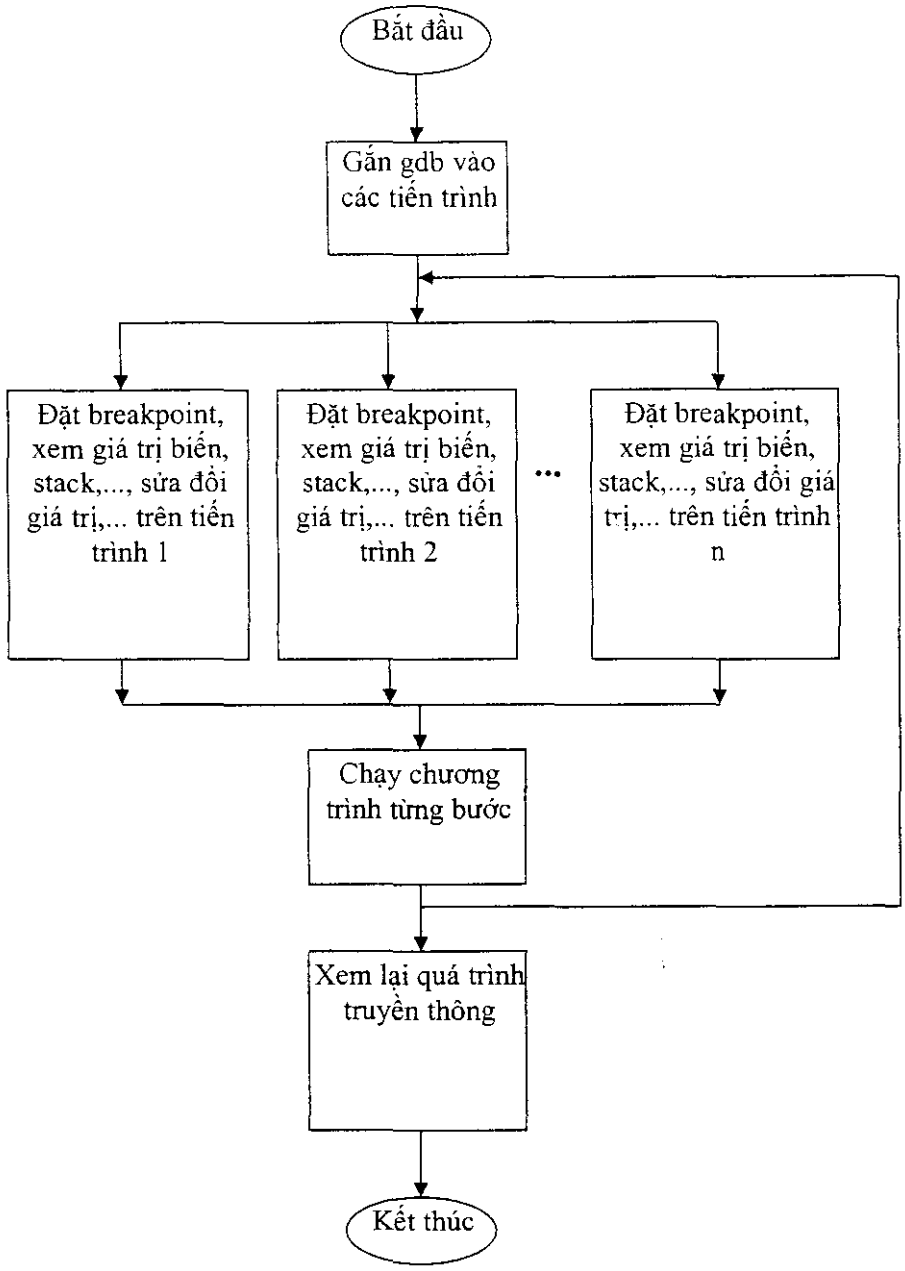
Mục tiêu hiện nay của dự án BKPD chưa phải là phát triển một chương trình hỗ trợ gỡ rối hoàn thiện, mà chỉ là phát triển một chương trình khung hỗ trợ gỡ rối một cách cơ bản. Để phát triển BKPD thành một chương trình gỡ rối song song hoàn thiện cần nhiều công nghiên cứu và phát triển hơn nữa.

3.2.4 Hoạt động của BKPD

Quá trình hoạt động của BKPD có thể được mô tả bước đầu như sau:

- Người sử dụng khởi động chương trình gỡ rối, kết nối đến server
- Server sẽ khởi động gdb trên các nút, gắn gdb vào các tiến trình song song cần chẩn đoán lỗi.
- Toàn bộ các tiến trình dừng lại, chờ các điều khiển của người sử dụng.
- Người sử dụng có thể thực hiện các hoạt động sau thông qua giao diện phía client.
    - switch giữa các tiến trình
    - đặt break point trên các tiến trình
    - xem giá trị các biến
    - xem giá trị stack
    - sửa đổi giá trị biến, stack, ô nhớ, thanh ghi
- Người dùng có thể cho chương trình tiếp tục chạy cho đến break point tiếp theo.
- Người dùng cũng có thể thực hiện chạy chương trình theo từng bước.
- Cuối cùng, người sử dụng có thể xem lại quá trình truyền thông của chương trình.

Sơ đồ tổng quan các bước trong quá trình debug

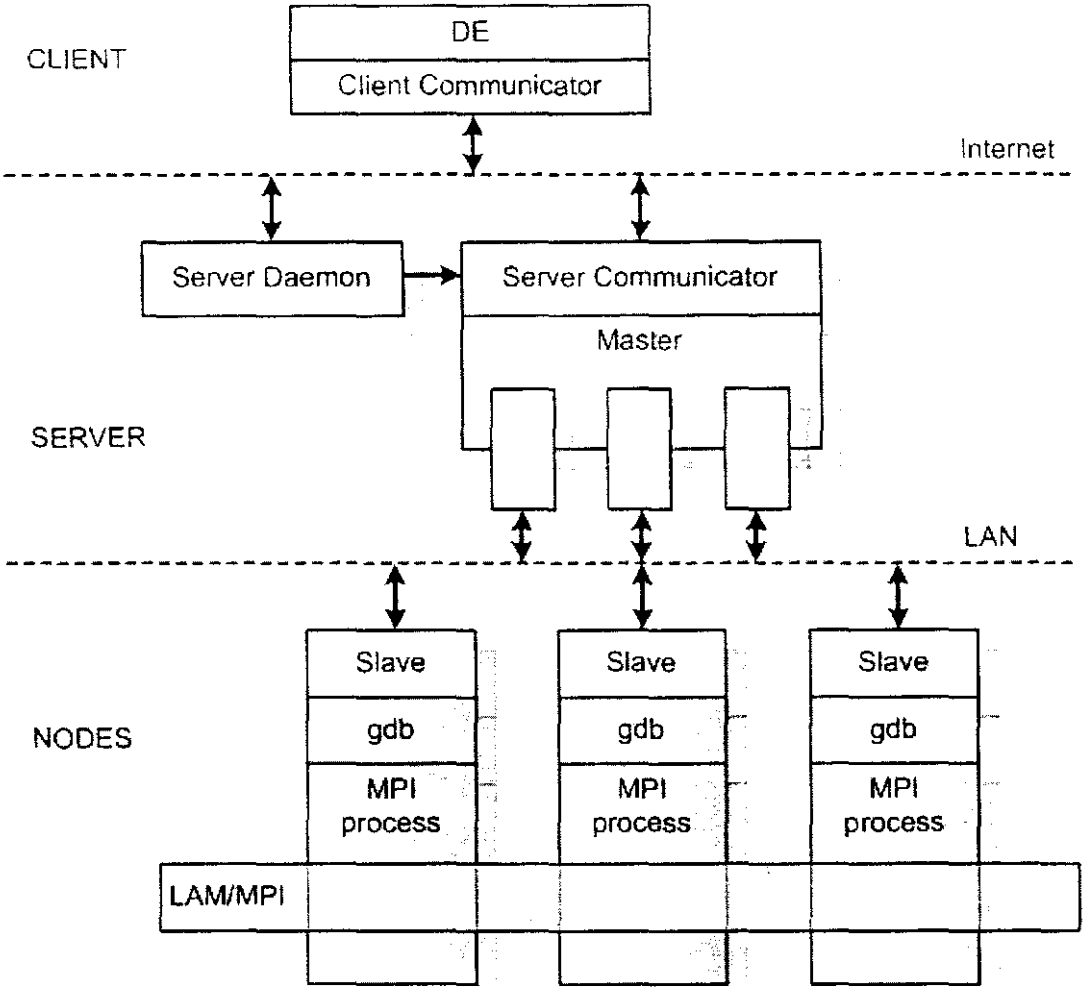


Hình 3-2 Các bước debug chương trình song song

3.3 KIẾN TRÚC BKPD

3.3.1 Kiến trúc chung

BKPD có mô hình kiến trúc theo các tầng như sau.



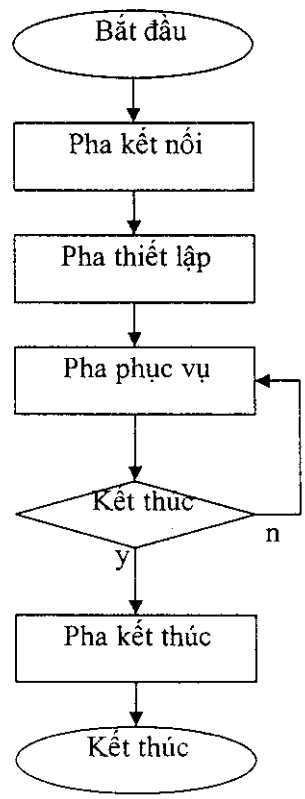
Hình 3-3 Mô hình kiến trúc BKPD

- DE (Debugging enviroment) : Môi trường gỡ lỗi song song. Tầng này nằm phía client, cung cấp cho người dùng khả năng tiến hành gỡ lỗi từ xa.
- Tầng truyền thông kết nối: bao gồm Client communicator, Server Daemon, và Server communicator.
- Client và Server communicator: Là tầng đệm trung gian đảm nhận nhiệm vụ truyền thông giữa client và server. Client communicator cung cấp các hàm API cho DE để thực hiện các câu lệnh debug. Nhờ Client và Server comm, DE nhìn hệ thống phân tán của ta giống như là hệ thống cục bộ.

- Server Daemon: Tiến trình ngầm phục vụ phía Server tính toán. Tiến trình này luôn chờ ở một cổng, mỗi khi có người dùng kết nối đến sẽ sinh ra tiến trình phục vụ.
- Master : Thành phần đảm nhận các công việc sau :
  - Khởi tạo các slave tại thời điểm ban đầu.
  - Là nơi xử lý lệnh nhận được từ người sử dụng, thực hiện lệnh, nhận kết quả trả về và xử lý kết quả (trước khi kết quả được truyền về cho người sử dụng).
- Slave : Thành phần điều khiển gdb hoạt động : truyền lệnh cho gdb, nhận kết quả trả về từ gdb và truyền cho master.
- Gdb.
- Chương trình MPI.

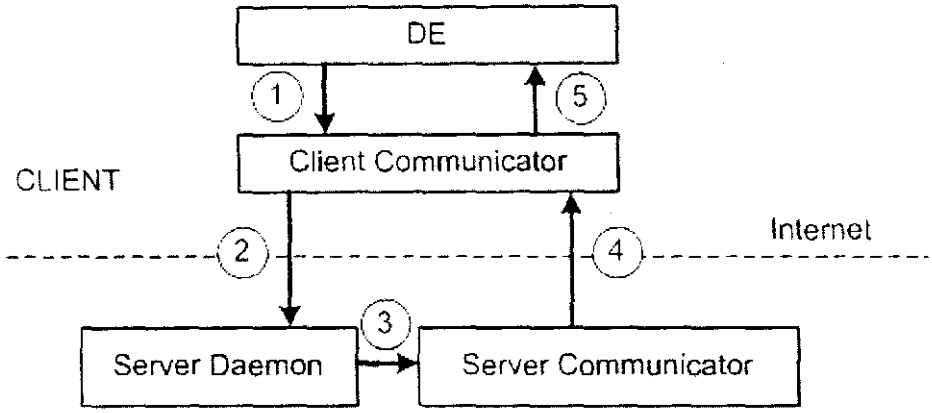
3.3.2 Kịch bản hoạt động của BKPD

Kịch bản hoạt động của BKPD bao gồm 4 pha chính : pha kết nối, pha thiết lập, pha phục vụ và pha kết thúc.



Hình 3-4 Kịch bản hoạt động của BKPD

3.3.2.1 Pha kết nối



Hình 3-5 Hoạt động của BKPD - Pha kết nối

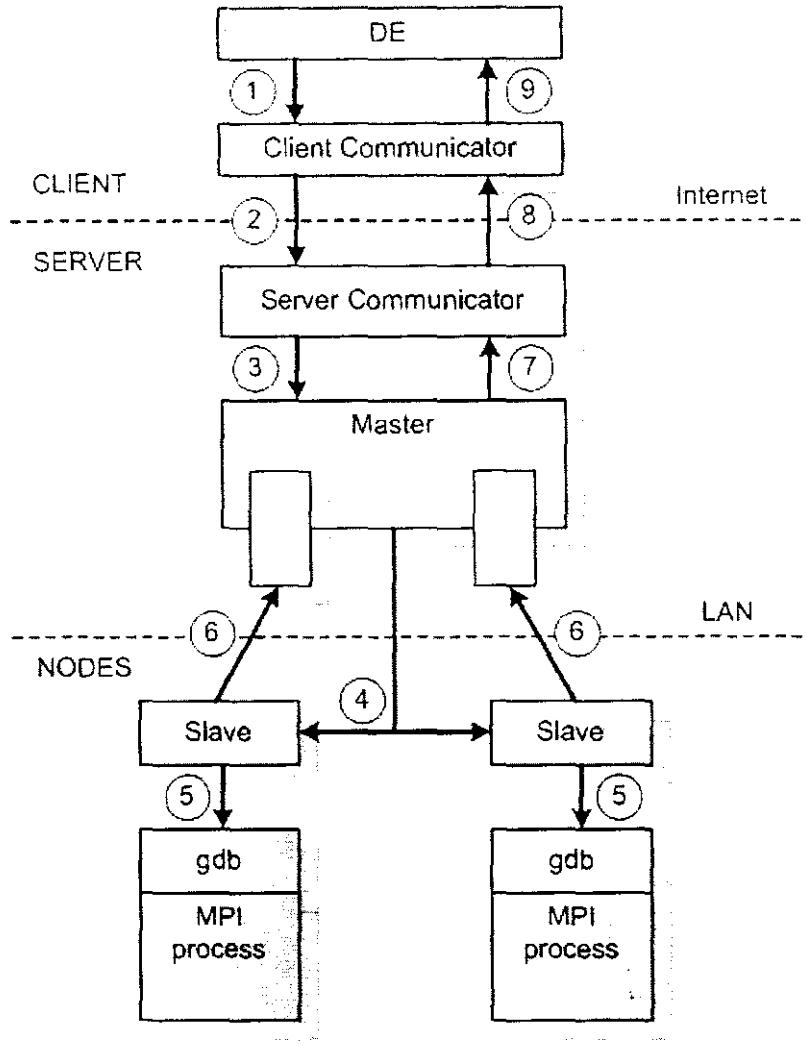
1. Người dùng sử dụng DE gọi các hàm của Client Communicator kết nối đến server.
2. Client Communicator kết nối đến server.
3. Server Daemon xác nhận người dùng, sau đó sinh ra tiến trình Server Communicator phục vụ người dùng.
4. Server Communicator gửi thông tin kết nối thành công về cho Client Communicator. Thông tin xác nhận chính là số nút tính toán có tại trung tâm.
5. Client Communicator thông báo về cho người dùng thông qua DE.

Lúc này hệ thống sẵn sàng phục vụ người dùng.

Các ngoại lệ có thể xảy ra :

- Client Communicator không kết nối được đến server (vấn đề đường truyền mạng)
- Server xác nhận người sử dụng không có quyền.

3.3.2.2 Pha thiết lập



Hình 3-6 Hoạt động của BKPD - Pha thiết lập

Trạng thái:

Client và Server Communicator đã kết nối, sẵn sàng chờ nhận yêu cầu của người dùng : tên chương trình cần chạy, số nút chạy.

Các bước :

1. Người sử dụng thông báo chương trình muốn gỡ lỗi và số nút chạy cho Client Communicator thông qua DE.
2. Client Communicator truyền thông tin đến Server Communicator.
3. Server Communicator khởi tạo Master, sau đó gọi hàm API của Master yêu cầu khởi tạo môi trường gỡ lỗi.
4. Master triệu gọi các Slave trên các nút tính toán.

5. Các Slave gọi gdb, khởi sinh các tiến trình MPI song song.
6. Slave thiết lập kết nối đến Master và gửi thông tin thông báo khởi tạo thành công về cho Master.
7. Master nhận thông báo từ slave, sau đó trả về kết quả cho Server Communicator.
8. Server Communicator truyền kết quả về cho Client Communicator.
9. Client Communicator trả về cho DE để hiển thị thông báo khởi tạo môi trường thành công.

Các ngoại lệ có thể xảy ra :

- Người dùng yêu cầu chạy chương trình không tồn tại.
- Các Slave khởi tạo không thành công (có vài Slave không chạy).

3.3.2.3 Pha phục vụ

Trạng thái ban đầu:

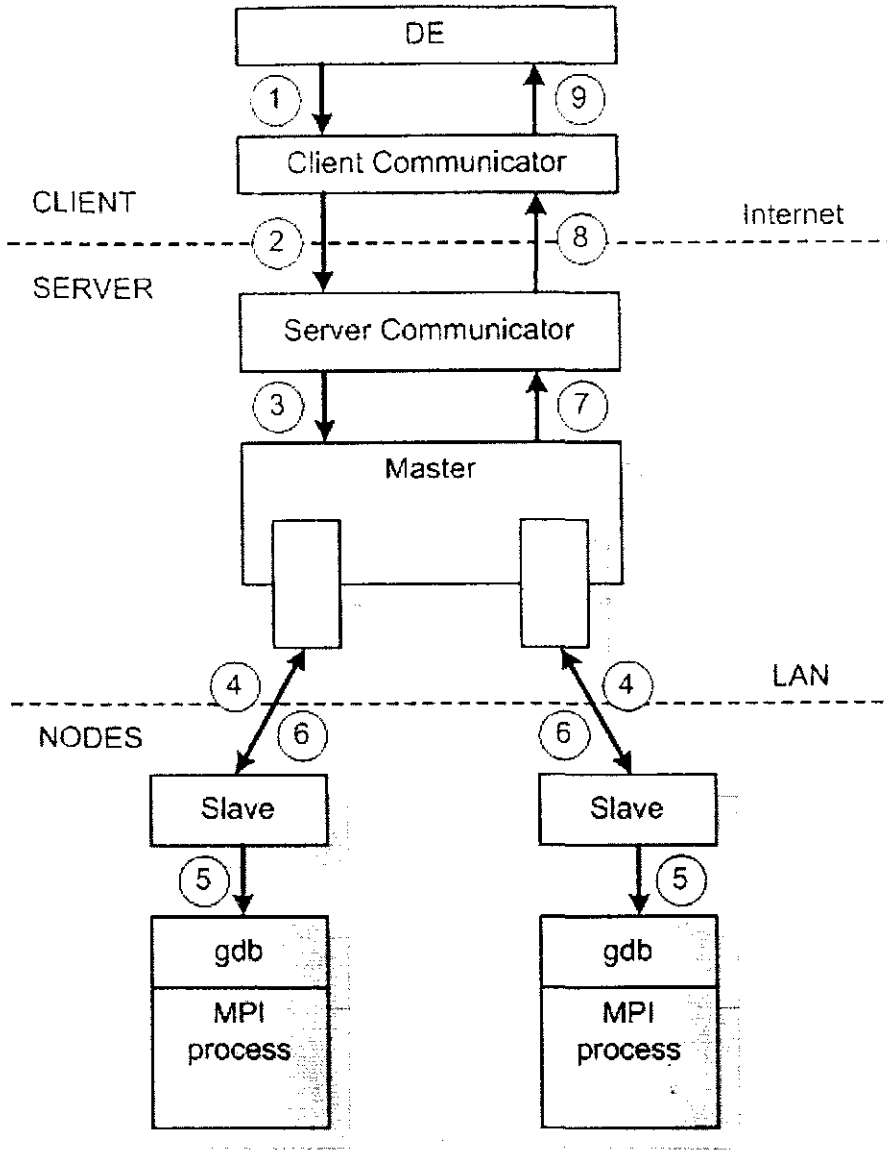
Môi trường debug đã được khởi tạo thành công. Hệ chờ người sử dụng nhập vào các lệnh debug.

Các bước :

1. DE gọi hàm thực hiện lệnh của người dùng do Client Communicator cung cấp.
2. Client Communicator truyền lệnh cho Server Communicator.
3. Server Communicator gọi hàm thực hiện lệnh do Master cung cấp.
4. Master xử lý lệnh, sau đó truyền xuống cho slave thông qua kết nối được thiết lập trong pha thiết lập.
5. Slave điều khiển gdb thực hiện lệnh.
6. Slave gửi kết quả về cho master.
7. Master tổng hợp kết quả trả về cho Server Communicator.
8. Server Communicator truyền kết quả về cho Client Communicator.
9. Client Communicator trả về kết quả cho DE.

Các ngoại lệ có thể xảy ra

- Câu lệnh sai cú pháp.
- Có một Slave bị block.



Hình 3-7 Hoạt động của BKPD - Pha phục vụ

3.3.2.4 Pha kết thúc

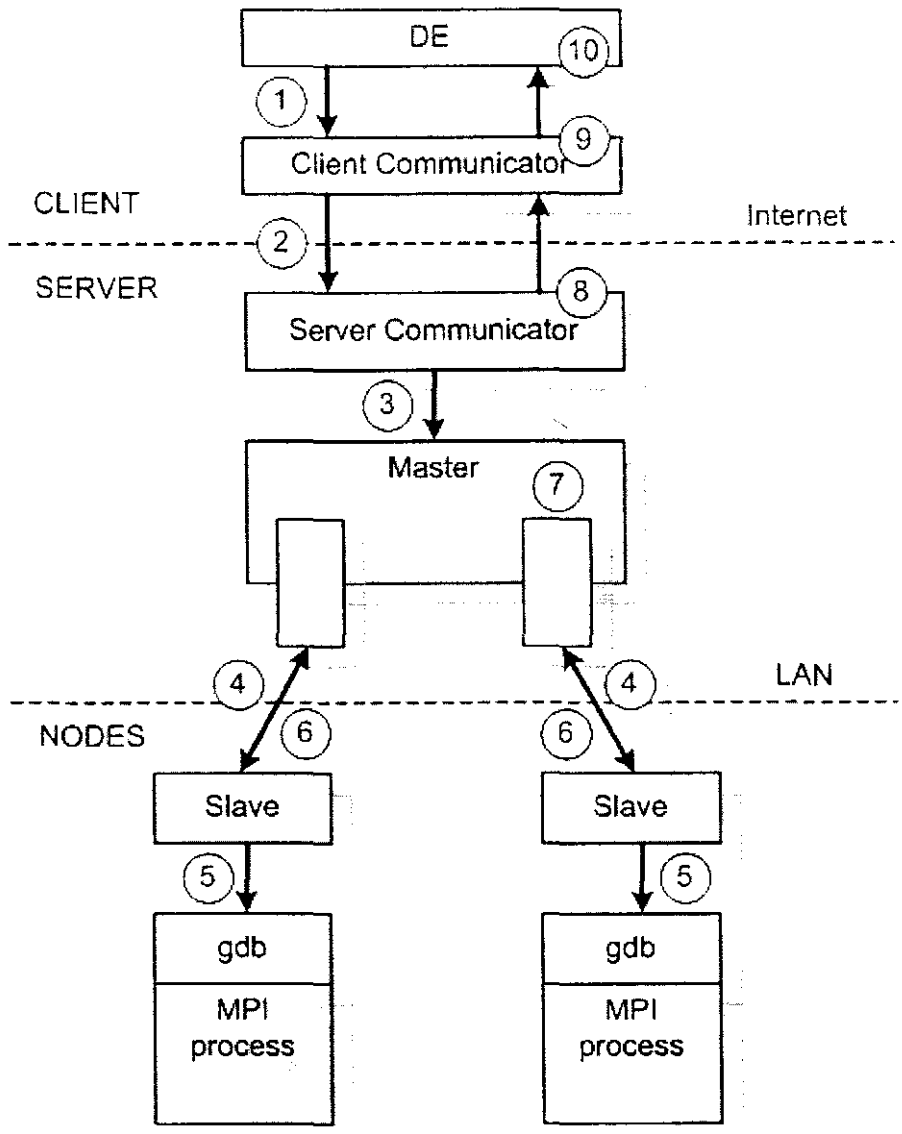
Trạng thái ban đầu:

Hệ đang hoạt động bình thường.

Các bước:

1. DE gọi hàm kết thúc của Client Communicator.
2. Client Communicator gửi thông báo kết thúc cho Server Communicator.
3. Server Communicator gọi hàm kết thúc của Master.
4. Master gửi thông báo kết thúc cho các Slave.
5. Các Slave điều khiển các gdb kết thúc phiên làm việc.

6. Slave gửi thông báo về cho Master và tự kết thúc.
7. Master tự kết thúc.
8. Server Communicator gửi thông báo về cho Client Communicator rồi tự kết thúc.
9. Client Communicator tự kết thúc.
10. DE kết thúc phiên phục vụ.



Hình 3-8 Hoạt động của BKPD - Pha kết thúc

3.3.3 Giao thức

Trong BKPD có hai giao tiếp mạng là giao tiếp giữa Client Communicator với Server và Server Communicator, và giao tiếp giữa Master và Slave. Dữ liệu trong

hai giao tiếp này được tổ chức thành các xâu dữ liệu trước khi truyền dis (phương pháp "*data is string*")

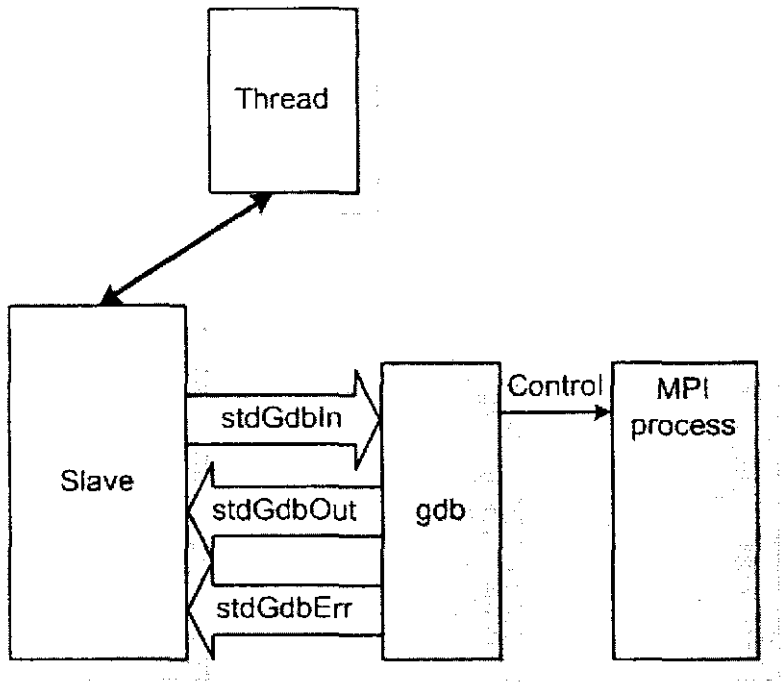
3.3.3.1 Thành phần Slave

Chức năng

Slave là thành phần chạy trên từng nút tính toán, đảm nhận vai trò điều khiển chương trình gỡ lỗi tuần tự gdb, thực hiện gỡ lỗi trên từng tiến trình riêng biệt của chương trình song song.

Các Slave sẽ gắn gdb vào tiến trình song song, nhận lệnh gỡ lỗi từ Master, điều khiển gdb thực hiện lệnh, sau đó trả kết quả về cho Master.

Kiến trúc



Hình 3-9 Thành phần Slave

Slave giao tiếp với gdb thông qua cơ chế đường ống pipe. Giao tiếp này bao gồm ba đường ống chuẩn sau:

- stdGdbIn: đường ống vào chuẩn của gdb.
- stdGdbOut: đường ống ra chuẩn của gdb.
- stdGdbErr: đường ống ra lỗi chuẩn của gdb.

Hoạt động

- Pha thiết lập

- Slave được khởi động bởi Master (câu lệnh "mpirun –np nnodes Slave mpi_prog port").
- Slave khởi động gdb.
- Slave gửi tên file chương trình song song cho gdb.
- Slave gửi thông báo xác nhận cho Master.

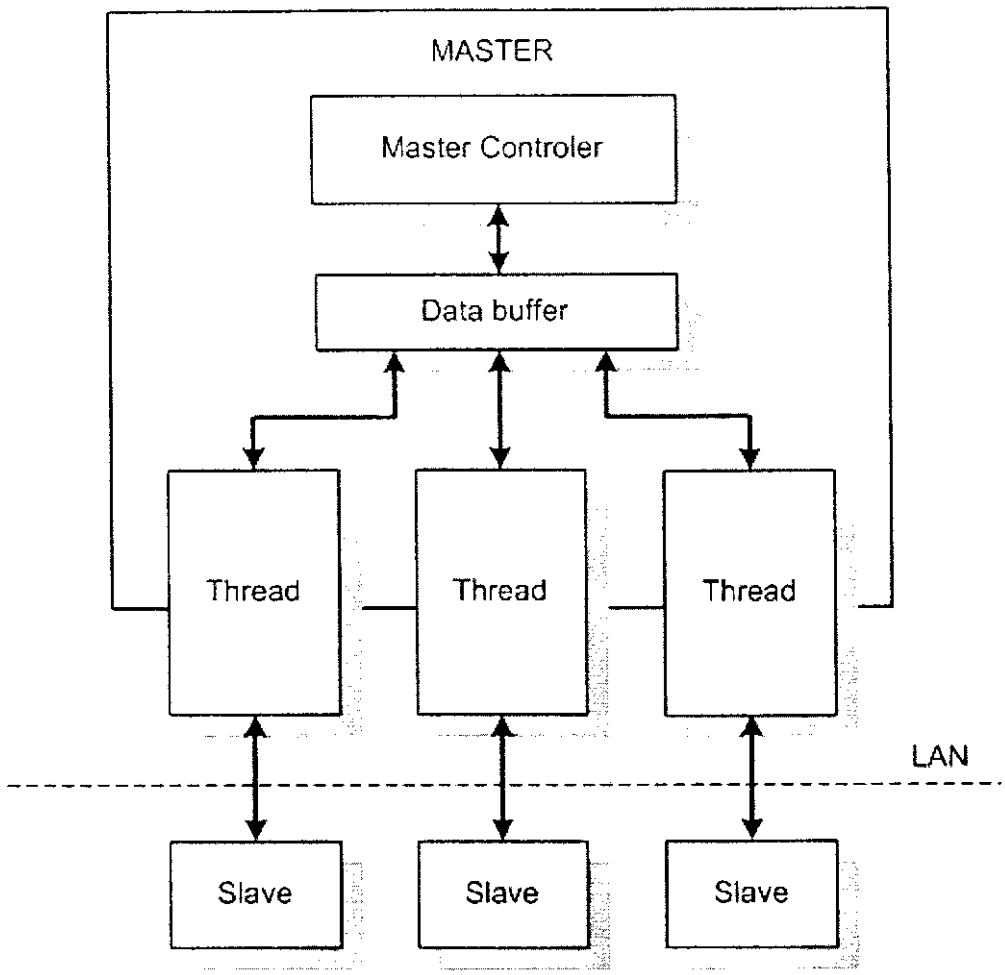
- Pha phục vụ
  - Slave nhận lệnh từ Master.
  - Slave đẩy câu lệnh vào stdGdbIn truyền cho gdb.
  - Slave chờ đọc kết quả trả về từ gdb trong đường ống stdGdbOut và thông báo lỗi trong đường ống stdGdbErr.
  - Slave truyền xâu kết quả về cho Master.
- Pha kết thúc
  - Slave nhận lệnh kết thúc từ Master.
  - Slave đẩy lệnh kết thúc vào stdGdbIn.
  - Slave gửi thông báo kết thúc cho Master.
  - Slave kết thúc.

3.3.3.2 Thành phần Master

Chức năng

- Thành phần Master đảm nhiệm vai trò điều khiển các nút tính toán thực hiện câu lệnh debug của người sử dụng.
- Master nhận lệnh từ Server, tiền xử lý lệnh, sau đó truyền lệnh cho các slave thông qua các thread.
- Master nhận kết quả trả về từ các thread, tổng hợp và trả về thông báo cho Server Communicator.

Kiến trúc



Hình 3-10 Thành phần Master

Master Controler

Đây là thành phần điều khiển chính của Master. Thành phần này đảm nhận trách nhiệm:

- Xử lý câu lệnh được chuyển tới.
- Truyền câu lệnh vào bộ đệm lệnh.
- Tác động lên cờ lệnh.
- Đọc dữ liệu kết quả trả về.
- Xử lý kết quả và thông báo cho Server Communicator.

Thread

Các tiến trình chạy độc lập có vai trò giao tiếp với các slave trên các nút tính toán. Thread thực hiện các công việc sau:

- Nhận lệnh từ bộ đệm lệnh.
- Truyền lệnh cho slave của mình.
- Nhận kết quả trả về từ slave.
- Ghi kết quả vào bộ đệm kết quả.
- Tác động vào cờ kết quả.

Data Bufer

Là trung gian lưu trữ dữ liệu giao tiếp giữa Master Controler và các Thread. Data Buffer bao gồm:

- Một cờ lệnh (command flag) để Master Controler có thể báo hiệu cho các Slave khi có lệnh
- Một vùng đệm lệnh (command buffer) lưu trữ câu lệnh cần thực hiện.
- Một vùng đệm kết quả (result buffer) lưu trữ các kết quả trả về của các Slave.
- Một cờ kết quả (result flag) để báo cho Master Controler biết khi nào một Slave thực hiện xong lệnh và có kết quả trả về.

Hoạt động

Hoạt động của Master thuộc vào ba pha chính: thiết lập, phục vụ và kết thúc. Tuy nhiên ta có thể coi pha kết thúc là pha phục vụ với câu lệnh thực hiện là câu lệnh kết thúc phiên làm việc.

Pha phục vụ bao gồm hai giai đoạn: nhận và thực hiện lệnh + xử lý và trả về kết quả.

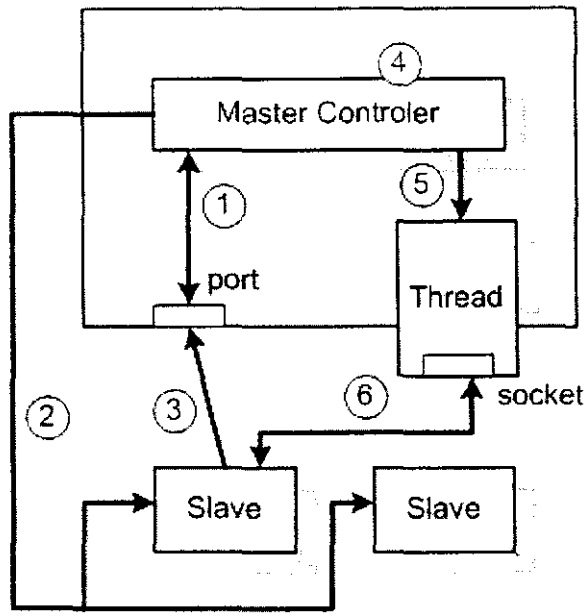
Thiết lập

Các bước hoạt động của Master khi thiết lập được mô tả như hình vẽ trên. Các hoạt động trong pha này được bắt đầu khi Master nhận được yêu cầu khởi tạo môi trường gỡ lỗi.

1. Khi nhận được yêu cầu thiết lập môi trường, đầu tiên Master Controler sẽ tạo ra một socket tại cổng port để chờ tin confirm của các Slave. Cổng port này sẽ được thay đổi tuỳ theo từng phiên làm việc để đảm bảo không có sự trùng lặp.
2. Master Controler sẽ triệu gọi các chương trình Slave trên các nút tính toán. Việc triệu gọi này được thực hiện thông qua câu lệnh mpirun: "mpirun –np nnodes Slave port mpi_prog". Nnodes (số nút chạy) và mpi_prog (tên chương trình chạy) được truyền từ client đến thông qua Client

Communicator và Server Communicator. Tham số port cho Slave biết cần gửi tin xác nhận đến cổng nào của Master.

3. Sau khi chạy thành công gdb và chương trình mpi trên nút tính toán của mình, Slave gửi thông tin xác nhận về cổng đã được thông báo.
4. Master Controler cập nhật thông tin về Slave.
5. Master Controler sinh ra một thread đảm nhận việc giao tiếp với Slave vừa confirm và một socket vô danh giao tiếp với Slave đó.
6. Sau đó thread vừa tạo ra sẽ có trách nhiệm giao tiếp với Slave thông qua socket vô danh này.



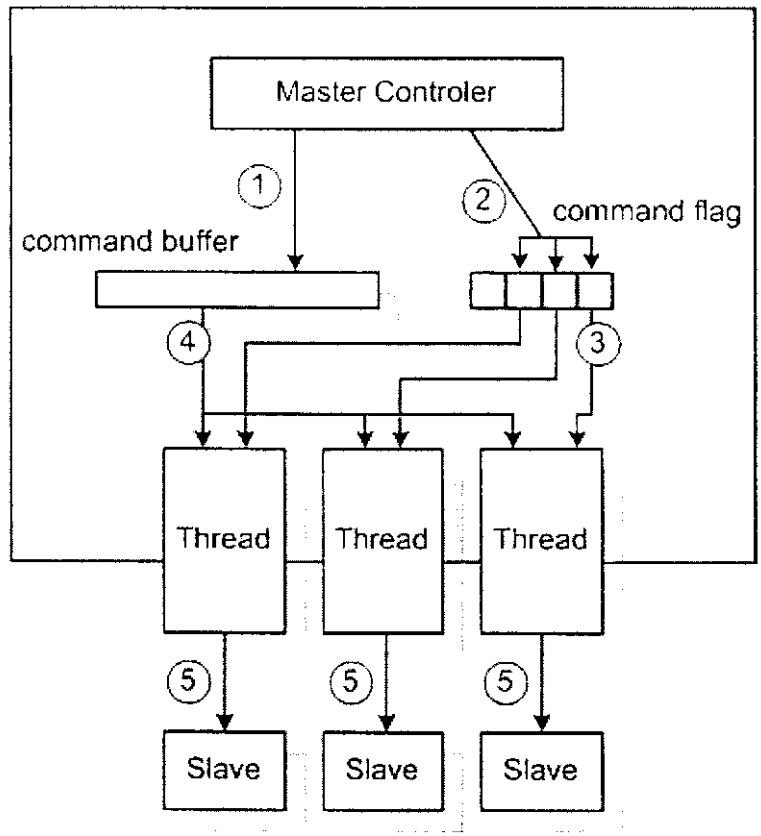
Hình 3-11 Hoạt động của master khi khởi tạo

Nhận và thực hiện lệnh

Trạng thái ban đầu: Hệ thống đã khởi tạo thành công. Người dùng yêu cầu thực hiện câu lệnh debug và câu lệnh được chuyển đến Master.

1. Master Controler đưa câu lệnh vào vùng đệm lệnh.
2. Master Controler set các cờ lệnh của các Slave cần thực hiện lệnh tương ứng.
3. Các thread liên tục đọc cờ lệnh tương ứng của mình để biết khi nào có lệnh.
4. Khi có lệnh, thread sẽ đọc câu lệnh trong vùng đệm lệnh và reset cờ lệnh của mình.

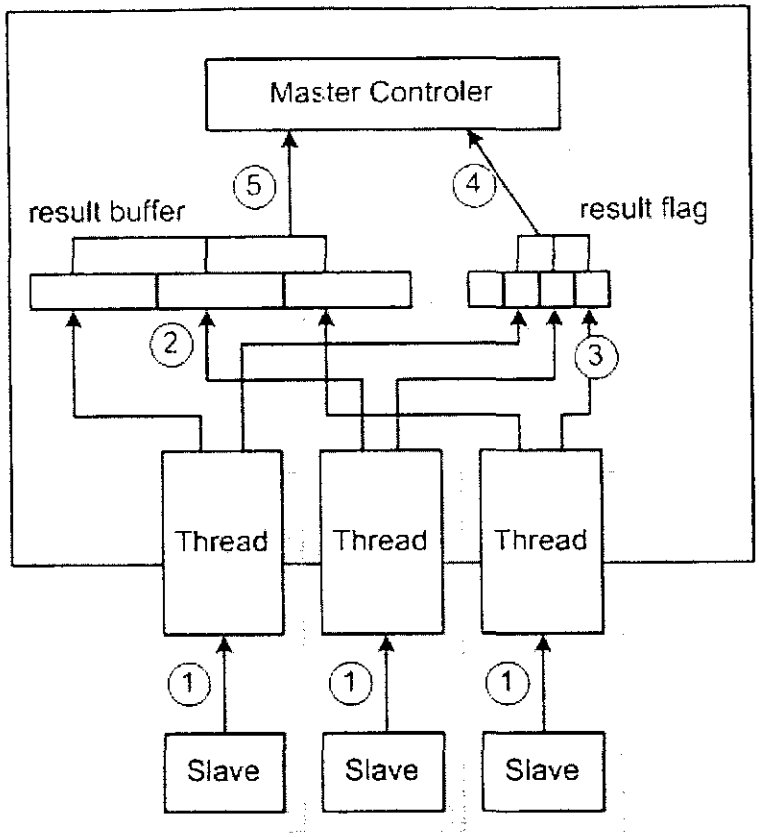
5. Thread truyền lệnh đến cho Slave.



Hình 3-12 Master nhận và thực hiện lệnh

Nhận và xử lý kết quả trả về

1. Các Slave thực hiện xong công việc sẽ gửi kết quả về cho các thread tương ứng.
2. Các thread ghi kết quả trả về vào vùng đệm dữ liệu tương ứng của mình.
3. Các thread set cờ kết quả tương ứng của mình.
4. Master Controler kiểm tra xem cờ kết quả của các Slave thực hiện lệnh đã được set chưa.
5. Khi tất cả cờ của các Slave thực hiện lệnh đã được set, khi đó Master Controler sẽ đọc và xử lý dữ liệu trong bộ đệm kết quả, đồng thời reset lại cờ kết quả.



Hình 3-13 Master nhận và xử lý kết quả

3.3.3.3 Thành phần Server Communicator

Chức năng

- Server Communicator là thành phần trung gian phía Server đảm nhận trách nhiệm giao tiếp với Client Communicator phía Client. Server Communicator nhận lệnh truyền đến từ phía client, gọi các hàm do Master cung cấp để thực hiện lệnh, sau đó trả kết quả về cho Client.
- Mỗi một người sử dụng khi yêu cầu một phiên làm việc sẽ có một tiến trình Server Communicator tương ứng phía Server phục vụ.
- Nhờ có Server Communicator và Client Communicator, DE phía client sẽ nhìn toàn hệ thống như là một chương trình cục bộ.

3.3.3.4 Thành phần Server Daemon

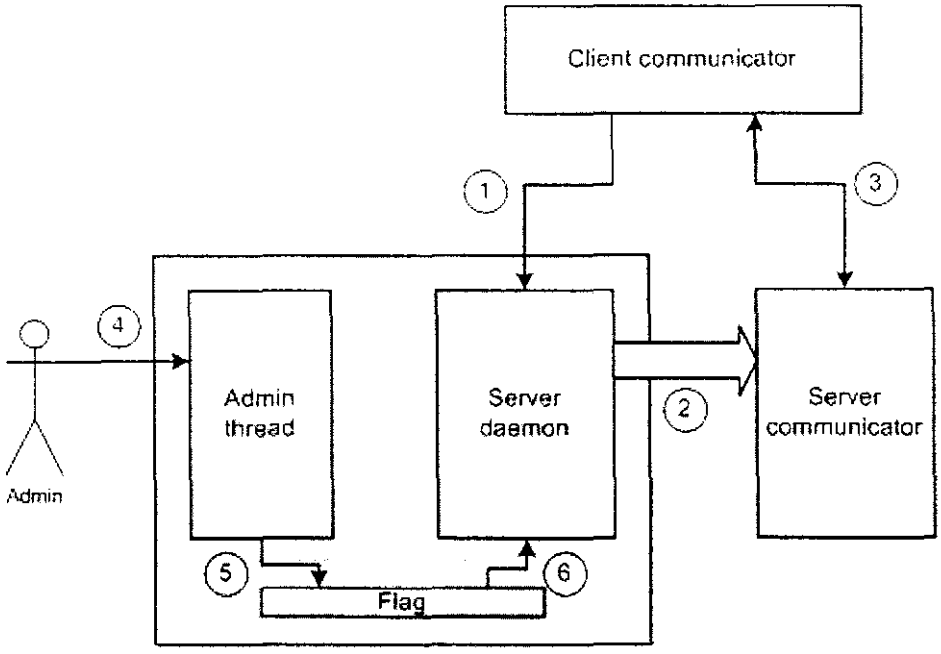
Chức năng

- Server Daemon là tiến trình ngầm chờ nhận kết nối của người dùng. Mỗi khi có người dùng mới, Server Daemon sẽ sinh ra một tiến trình mới phục vụ.
- Server Daemon cũng sinh ra một thread riêng cho phép người quản trị có khả năng can thiệp vào quá trình hoạt động của hệ thống như xem số người

sử dụng, khoá và mở khoá server...

- Server Daemon hiện nay mới chỉ là một module đơn giản, trong tương lai có thể phát triển hơn hoặc có thể sẽ bị thay thế bởi server chung của cả hệ thống BKluster.

Kiến trúc



Hình 3-14 Thành phần server daemon

Trong thành phần Server Daemon bao gồm ba thành phần chính:

- Server Daemon: là thành phần chính, tiến trình ngầm chờ và phục vụ người sử dụng.
- Admin thread: là thread chạy song song, cho phép admin quản trị hoạt động của Server.
- Flag: các cờ để báo hiệu các lệnh của người dùng đến Server daemon.

Hoạt động

1. Client communicator kết nối đến Server daemon thông qua một cổng định sẵn (12000).
2. Server daemon sinh ra tiến trình Server communicator phục vụ người sử dụng, đồng thời cũng cập nhật các thông tin về người sử dụng hệ thống.
3. Phiên làm việc được thực hiện thông qua giao tiếp giữa Client communicator và Server communicator.

4. Người sử dụng điều khiển hoạt động của Server thông qua Admin thread.
5. Khi nhận được lệnh của admin, Admin thread sẽ cập nhật vào cờ lệnh tương ứng.
6. Server Daemon nhận lệnh thông qua cờ lệnh và thực hiện lệnh.

Hiện nay hệ thống mới chỉ hỗ trợ một số lệnh đơn giản như: xem số người dùng (list), hạn chế không cho thêm người sử dụng dùng hệ thống (lock), xoá bỏ hạn chế (unlock), kết thúc (q).

3.3.3.5 Thành phần Client Communicator

Chức năng

- Client Communicator cùng với Server Communicator tạo thành bộ đệm trung gian đảm nhận vai trò truyền thông giữa Client và Server.
- Client communicator cung cấp một tập hàm API cho DE sử dụng để thực hiện câu lệnh debug.

Tập các hàm API Client communicator cung cấp

- `int Init(QString serverIP,int port,QString userID,int Dtimeout);`

  Là hàm kết nối đến Server daemon.
- `int Running(QString mpiProg, int nnodes);`

  Là hàm khởi tạo môi trường debug.
- `int Execute(QString userCommand);`

  Là hàm thực thi lệnh debug.
- `QString getResult(int rank);`

  Là hàm đọc kết quả trả về.

3.3.3.6 Thành phần BKViewer

BKViewer và một số các phầm mềm theo dõi truyền thông khác như Xmpi trực quan hoá quá trình truyền thông bằng cách sử dụng dữ liệu từ file lưu vết truyền thông điệp (trace file) trong môi trường LAM. File vết truyền thông này được LAM tạo ra trong quá trình chạy chương trình song song.

Tạo file vết truyền thông

Để tạo ra trace file, ta cần thực hiện hai bước :

1. Chạy chương trình song song với tuỳ chọn tạo trace file (-t):

   mpirun –np nnodes –t mpiProg

Khi chạy chương trình với tuỳ chọn này, các lam daemon trên các nút tính toán sẽ lưu lại các vết truyền thông điệp trên nút của mình.

2. Tổng hợp các vết truyền thông trên các nút tính toán thành một tracefile duy nhất sử dụng lệnh lamtrace :

lamtrace -k -mpi traceFileName

- Lựa chọn –k : không xoá các trace trên từng nút sau khi tổng hợp xong.
- Lựa chọn –mpi : chỉ tổng hợp trace của chương trình song song

Lệnh lamtrace sẽ tổng hợp các trace của chương trình mpi chạy gần nhất.

Khi đã có trace file, ta cần đọc dữ liệu dữ liệu để hiện thị lên đồ hoạ. Nội dung trace file được chia làm ba phần chính như sau :

- Vết môi trường (world trace) bao gồm số hiệu xác nhận format (magic number) và danh sách mô tả các nút tham gia chạy chương trình song song.
- Vết đối tượng (object trace) mô tả các đối tượng MPI như các kiểu dữ liệu (datatype) và các cụm truyền thông (communicator).
- Vết thực thi (run-time trace) mô tả các hoạt động xảy ra khi chạy chương trình.

Các dữ liệu trong trace file đều được tổ chức dưới dạng cấu trúc của C (C structure). Các kiểu dữ liệu cơ bản được dùng là char, int4 (kiểu int 4 byte) và float8 (kiểu thực 8 byte).

Tất cả các trace ngoài world trace đều có một phần dữ liệu nằm ở đầu (source subtrace) cho biết trace đó thuộc loại nào.

World trace

- World trace là thành phần đầu tiên của trace file, bao gồm:
- magic number (int4).
- Kích thước của môi trường LAM (MPI_COMM_WORLD) (int4).
- Danh sách mô tả các tiến trình (Process Description) theo thứ tự rank trong MPI_COMM_WORLD.

  Cấu trúc Process Discription như sau:

```
struct _gps {
    int4 gps_node;
    Đây là số hiệu nút đang chạy tiến trình trong mạng.
    int4 gps_pid;
```

```
        Đây là số hiệu tiến trình (process id) trên nút đó.
        int4 gps_idx;
        Số hiệu của tiến trình trong chương trình.
        int4 gps_grank;
        Rank của tiến trình trong MPI_COMM_WORLD.
    };
```

Danh sách cấu trúc này sẽ cho ta biết thông tin chính xác về các tiến trình có mặt trong môi trường LAM.

Source subtrace

Các trace ngoài world trace đều có một cấu trúc nằm ở đầu (source subtrace) cho biết trace đó thuộc loại gì cũng như một số thông tin khác về trace. Cấu trúc của source subtrace như sau :

```
struct trsrc {
int4 trs_node; /* node of process that made trace */
int4 trs_pid; /* pid of process that made trace */
node và pid phải phù hợp với world trace.
int4 trs_listno;
Cho biết trace thuộc loại gì.
int4 trs_pad; /* for alignment */
};
```

Các giá trị có thể có của trs_listno là:

- TRCOMM (communicator object trace) (giá trị = -2)
- TRDTYPE (datatype object trace) (giá trị = -3)
- TRONOFF (run-time trace của lớp TRONOFF (sẽ được nói ở sau)) (giá trị = -4)
- TRRUNTIME (run-time trace of class TRRUNTIME (sẽ được nói ở sau)) (giá trị=0)

Object trace

Có hai loại object trace :

- communicator trace : mô tả về các communicator.
- datatype trace : mô tả các kiểu datatype do người sử dụng định nghĩa.

Cả hai loại object trace này đều có một source subtrace ở đầu mô tả các thông số

của trace.

Communicator trace

Trace này theo sau một source subtrace có kiểu TRDCOMM (-2). Cấu trúc communicator trace như sau :

```
struct trcid {
   int4 trc_cid; /* cid */
   Đây là số không âm định danh cho communicator. Các giá trị
   từ 0 đến 2 tương ứng được dành cho các communicator chuẩn
   MPI_COMM_WORLD, MPI_COMM_SELF và MPI_COMM_PARENT.
   int4 trc_nlg; /* size of local group */
   Kích thước của nhóm các nút nội bộ.
   int4 trc_nrg; /* size of remote group */
   Kích thước của nhóm từ xa.
   int4 trc_pad; /* for alignment */
};
```

Theo sau cấu trúc trcid là danh sach các cấu trúc mô tả tiến trình (Process Description) cho các nút trong nhóm nội bộ, và tiếp đến là các nút trong nhóm từ xa. Danh sách này được sắp xếp theo thứ tự của rank.

Datatype trace

Trace này theo sau source subtrace có kiểu TRDTYPE (-3). Mỗi datatype trace bao gồm một chuỗi các phần tử theo cấu trúc trdtype như sau :

```
struct trdtype {
   int4 trd_param0;
   int4 trd_param1;
};
```

Giá trị các trường trong một phần tử trdtype còn tuỳ thuộc vào vị trí của phần tử đó trong chuỗi. Phần tử đầu tiên trong chuỗi chứa nhãn kiểu dữ liệu (datatype label) và độ dài của chuỗi (theo byte, bao gồm cả phần tử đầu tiên). Các phần tử tiếp theo trong chuỗi được sử dụng để mô tả kiểu dữ liệu được định nghĩa.

Run-time trace

Run-time trace bao gồm nhiều loại khác nhau. Có thể chia các loại này thành hai lớp chính : TRONOFF và TRRUNTIME. Thông qua source subtrace của một trace, ta có thể biết được trace đó thuộc TRONOFF hay TRRUNTIME.

Bên cạnh source subtrace, mỗi một trace trong run-time trace còn bao gồm một

phần tiêu đề nằm ngay sau source subtrace (run-time header). Phần tiêu đề này sẽ cho ta biết các thông tin cần thiết đặc trưng riêng của các run-time trace.

Run-time header

Phần tiêu đề có cấu trúc như sau:

```
struct trrthdr {
    float8 trr_time; /* trace time (sec) */
    Thời gian (tính theo giây). Đây là thời gian xảy ra sự kiện
    được mô tả trong run-time trace theo sau. Thời gian này sẽ
    được nói cụ thể trong từng run-time trace.
    int4 trr_type; /* trace type */ The flavor of the following
    trace.
    Loại của run-time trace theo sau.
    int4 trr_pad; /* for alignment */;
};
```

Giá trị của trr_type được liệt kê trong bảng sau:

Tên hằng số	Giá trị hằng số	Loại của run-time trace
TRTINIT	0	Khởi tạo
TRTSUBCHG	1	Thay đổi nền
TRTOUTPUT	2	Message đi ra
TRTINPUT	3	Message đi vào
TRTNOIO	4	Các loại message khác
TRTRON	5	Bật theo dõi
TRTROFF	6	Tắt theo dõi
TRTBUOY	7	
TRCOLORON	8	Bật màu
TRCOLOROFF	9	Tắt màu
TRCOMMNAME	10	Tên của communicator

Bảng 3-2 Các giá trị của struct trr_type

Chúng ta sẽ nói tỉ mỉ hơn về một số loại run-time trace trong các mục sau.

Lớp TRONOFF

- Lớp TRONOFF bao gồm các loại : Khởi tạo (TRTINIT), bật theo dõi

(TRTRON) và tắt theo dõi (TRTROFF).

- Các trace thuộc lớp TRONOFF đứng ngay sau world trace trong trace file.
- Trace khởi tạo (init trace) ghi lại các thông số của tiến trình tại thời điểm thực hiện câu lệnh MPI_Init. Trace có cấu trúc như sau:
  - Source subtrace có kiểu TRONOFF.
  - Tiêu đề (run-time header) có giá trị của trr_type là TRTINIT. Giá trị của trr_time trong tiêu đề chính là thời điểm thực hiện câu lệnh MPI_Init của tiến trình.

Cấu trúc trinit lưu thông tin riêng của init trace. Cấu trúc này như sau:

```
struct trinit {
   float8 tri_skew; /* clock skew (sec) */
   Độ lệch thời gian so với tiến trình có rank 0.
   char tri_name[32]; /* application name */
   Tên của chương trình thực thi.
};
```

Các trace bật theo dõi và tắt theo dõi mô tả thời điểm bắt đầu và kết thúc của quá trình theo dõi vết của chương trình song song. Quá trình theo dõi này là khoảng thời gian LAM sẽ theo dõi và ghi lại quá trình truyền thông của chương trình song song. Nếu quá trình theo dõi tắt, LAM sẽ không ghi lại các vết truyền thông nữa cho đến khi người lập trình bật lại truyền thông.

Người sử dụng có thể điều khiển bật và tắt quá trình theo dõi truyền thông này thông qua các câu lệnh API của thư viện MPI trong chương trình song song của mình. Đây là một hỗ trợ thuận lời giúp người lập trình có thể điều khiển theo dõi những phần truyền thông cần quan tâm, thay vì theo dõi toàn bộ chương trình. Và như vậy, một chương trình có thể bao gồm nhiều quá trình theo dõi rời nhau.

Theo mặc định, nếu người sử dụng không tác động gì đến việc bật/tắt theo dõi, LAM sẽ tự động theo dõi toàn bộ chương trình.

Cả hai trace bật và tắt theo dõi đều có format như sau:

```
struct tronoff {
   int4 tro_trnum; /* trace epoch number */
   Số hiệu định danh cho một quá trình theo dõi..
   int4 tro_pad; /* for alignment */
};
```

Lớp TRRUNTIME

Lớp TRRUNTIME mô tả các hoạt động xảy ra khi chạy chương trình song song, chủ yếu liên quan đến các hoạt động truyền thông. Lớp này gồm có một số loại như sau:

- TRTOUTPUT, TRTINPUT và TRTNOIO: dùng để thể hiện các truyền thông điểm-điểm (point to point) và các truyền thông quảng bá (collective communication). TRTOUPUT (output trace) chỉ một thông điệp đi ra từ một nút tính toán, và tương ứng với nó có một thông điệp đi vào (TRTINPUT) đi vào nút đích. TRTNOIO đại diện cho các thông điệp truyền thông quảng bá hoặc các trường hợp không có message nào được truyền (VD: MPI_Probe). TRTOUTPUT, TRTINPUT và TRTNOIO đều có format dạng cấu trúc trmsg.
- TRTRON và TRTROFF tương ứng với trong lớp TRONOFF. Các trace này cho biết hoạt động xảy ra trong đoạn theo dõi nào.

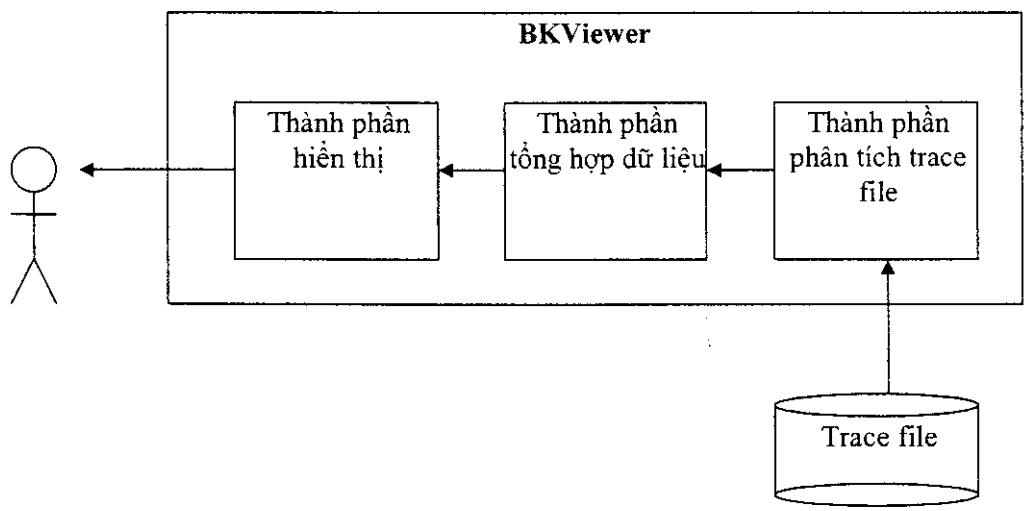
Cấu trúc `trmsg` như sau:

```
struct trmsg {
    int4 trm_topfunc; /* top-level function */
    int4 trm_wrapfunc; /* wrapper function */
    int4 trm_syst; /* trace time in system (usec) */
    Thời gian hoạt động này sử dụng CPU. Trong trường hợp truyền
    thông thì đây là thời gian thực hiện nhận/gửi thông điệp.
    int4 trm_blkt; /* trace time blocked (usec) */
    Thời gian block. Đây là khoảng thời gian tiến trình chờ nhận
    hoặc gửi thông điệp, là khoảng thời gian chết của chương
    trình.
    int4 trm_peer; /* peer rank */
    Sử dụng trong truyền thông điểm-điểm. Đây là rank của tiến
    trình giao tiếp với tiến trình hiện tại.
    int4 trm_tag; /* tag */
    Giá trị tag được sử dụng trong thông điệp truyền đi.
    int4 trm_cid; /* context ID */
    Số định danh communicator. Số định danh này cho biết message
    được truyền trong communicator nào.
    int4 trm_dtype; /* datatype label */
    Nhãn kiểu dữ liệu của message.
    int4 trm_count; /* message count */
```

```
    Số phần tử dữ liệu được truyền đi.

    Các giá trị tag, cid, dtype và count được chỉ ra trong câu
    lệnh MPI_Send của chương trình song song.

    int4 trm_mrank; /* src rank matched in case of wcard */

    int4 trm_mtag; /* tag matched in case of wcard */

    int4 trm_seqnum; /* sequence number */

    Giá trị seqnum (sequence number) là giá trị dùng để ghép
    khớp một thông điệp ra với một thông điệp vào. Mỗi một thông
    điệp ra (output message ) sẽ có một thông điệp vào tương ứng
    với nó với cùng số seqnum.
};
```

Kiến trúc BKViewer

BKViewer có kiến trúc như sau:



Hình 3-15 Kiến trúc BKViewer

- Thành phần phân tích trace file: có vai trò lọc ra các dữ liệu từ file trace.
- Thành phần tổng hợp dữ liệu: Tổng hợp các dữ liệu do thành phần phân tích trace file đọc, đưa chúng vào một cơ sở dữ liệu dễ xử lý hơn cho quá trình hiển thị.
- Thành phần hiển thị: Phân tích cơ sở dữ liệu và hiển thị trực quan lên màn

hình cho người sử dụng. Thành phần hiển thị bao gồm bốn module chính:

- Module hiển thị môi trường LAM (LAM World): Hiển thị danh sách các nút tính toán có trong môi trường LAM.
- Module hiển thị quá trình truyền thông (Message Passing Progress): Hiển thị trực quan quá trình truyền thông giữa các nút tính toán.
- Module hiển thị thông tin nút tính toán (Process Window): Hiển thị danh sách các thông điệp đến và đi ra từ một nút tính toán.
- Module hiển thị thông tin thông điệp (Message Window): Hiển thị thông tin chi tiết về thông điệp.

Trace file được truyền đến máy client thông qua giao thức truyền file FTP.

3.4 Một Số Kết Quả Đạt Được Và Định Hướng Phát Triển

3.4.1 Cài đặt hệ thống

Để cài đặt BKPD, hệ thống cần thoả mãn các yêu cầu sau:

- Có một mạng tính toán (một hay nhiều máy) cài đặt hệ điều hành Linux và môi trường LAM (phiên 7.0.6 trở lên).
- Có một máy front-end đảm nhận trách nhiệm giao tiếp với phía client. Đây chính là nơi chạy server của BKPD. Front-end cũng yêu cầu cài Linux và LAM. Máy front-end nên có cấu hình cao tương đối để có thể chạy server ổn đinh. Cấu hình đề nghị là: Pentium IV 1.2 GHz, RAM 256 MB hoặc hơn.
- Máy client có thể cài Linux hoặc Windows.

Việc cài đặt bao gồm hai bước:

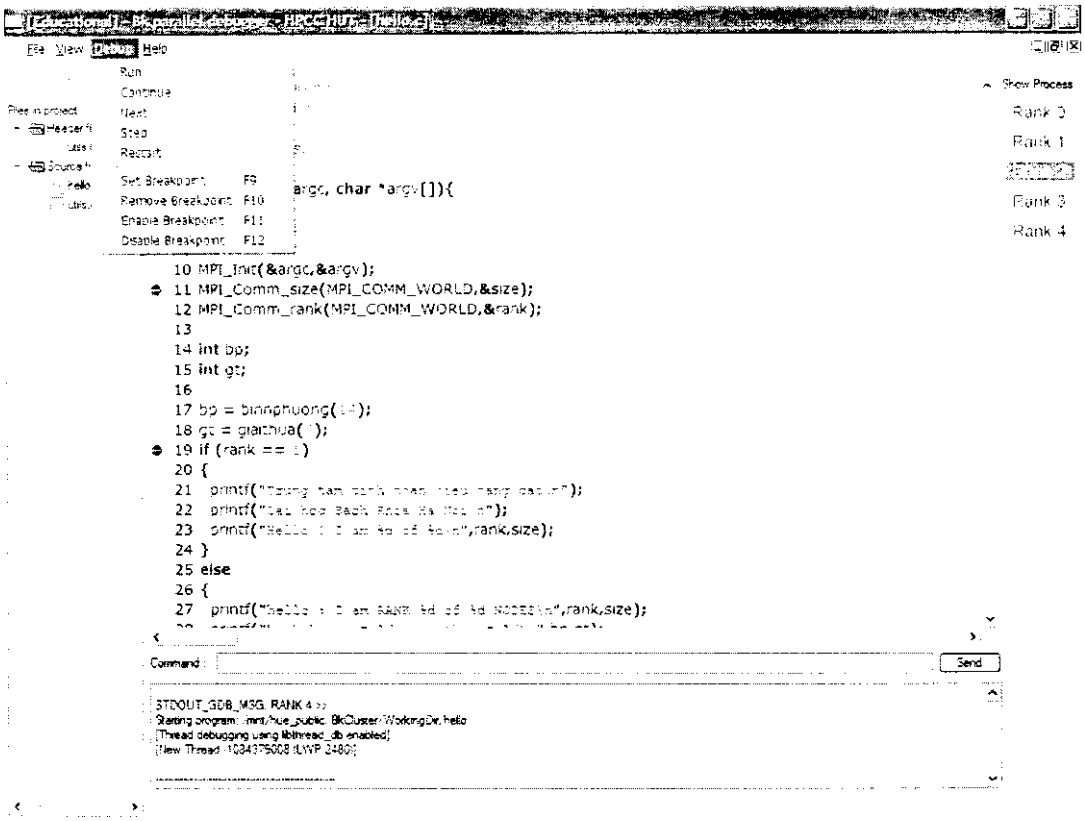
- Cài đặt server trên máy front-end. Có thể cài bằng source hoặc rpm.
- Cài đặt BKPD-client trên máy client.

Các thành phần của BKPD sau này sẽ được tích hợp vào gói phần mềm chung BKlusware. Người dùng có thể lựa chọn cài đặt BKPD khi cài gói phần mềm BKlusware.

3.4.2 Kết quả thử nghiệm

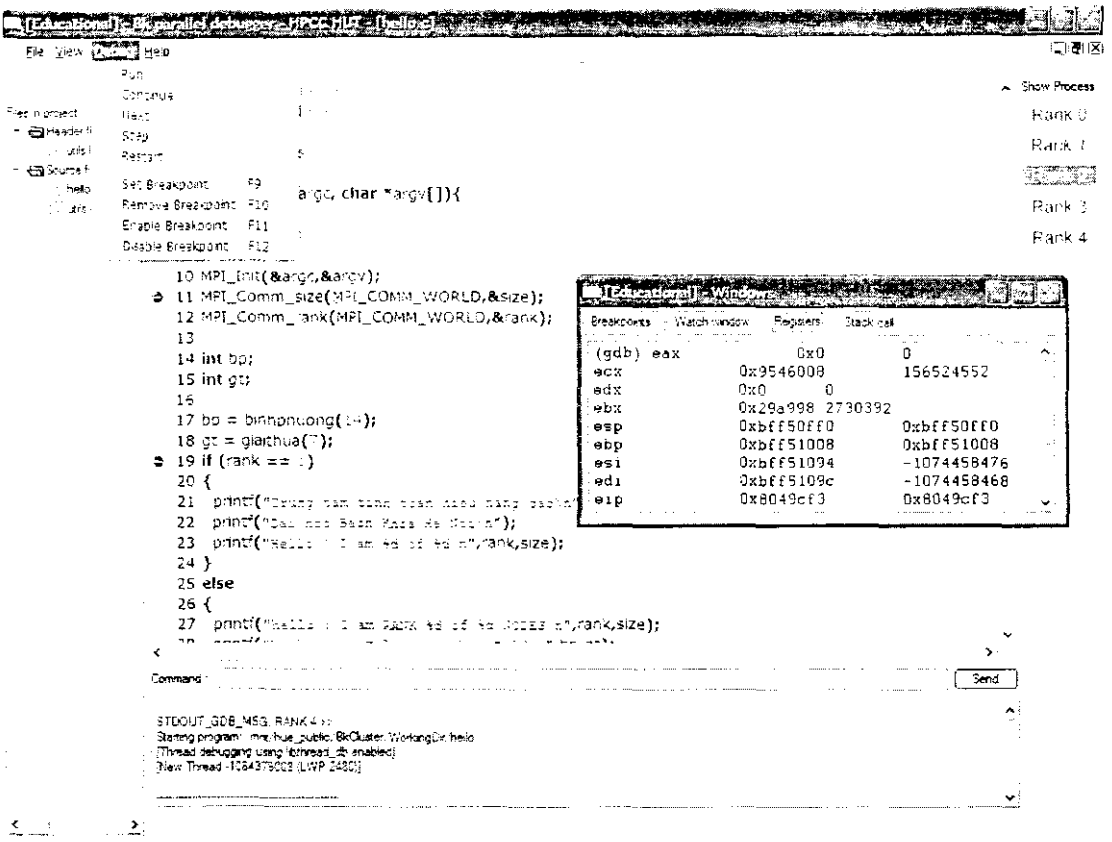
BKPD cung cấp hai khả năng chính: khả năng tương tác. điều khiển chương trình và khả năng theo dõi truyền thông điệp.

Người sử dụng có thể tương tác, điều khiển chương trình song song thông qua một giao diện đồ hoạ.



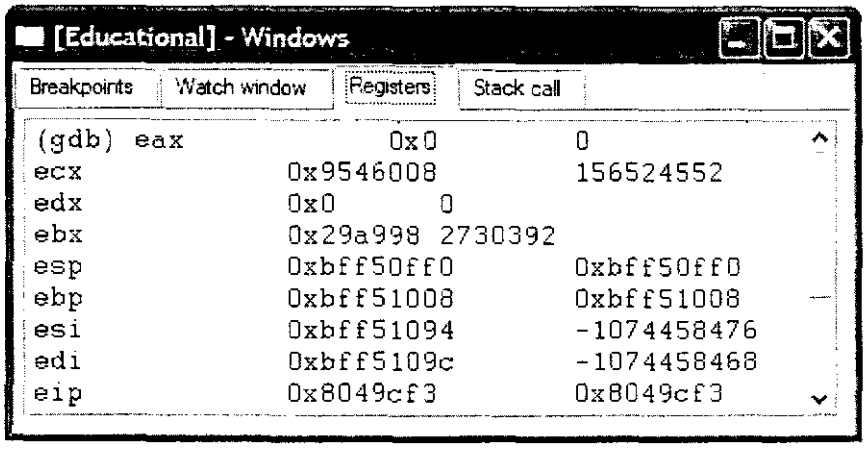
Hình 3-16 Cửa sổ chính của BKPD

Cửa sổ chính của chương trình cho phép người sử dụng có thể điều khiển tất cả các tiến trình cùng một lúc. Ngoài ra người sử dụng có thể sử dụng một cửa sổ dòng lệnh để nhập lệnh trực tiếp cho server mà không qua các thao tác với giao diện.



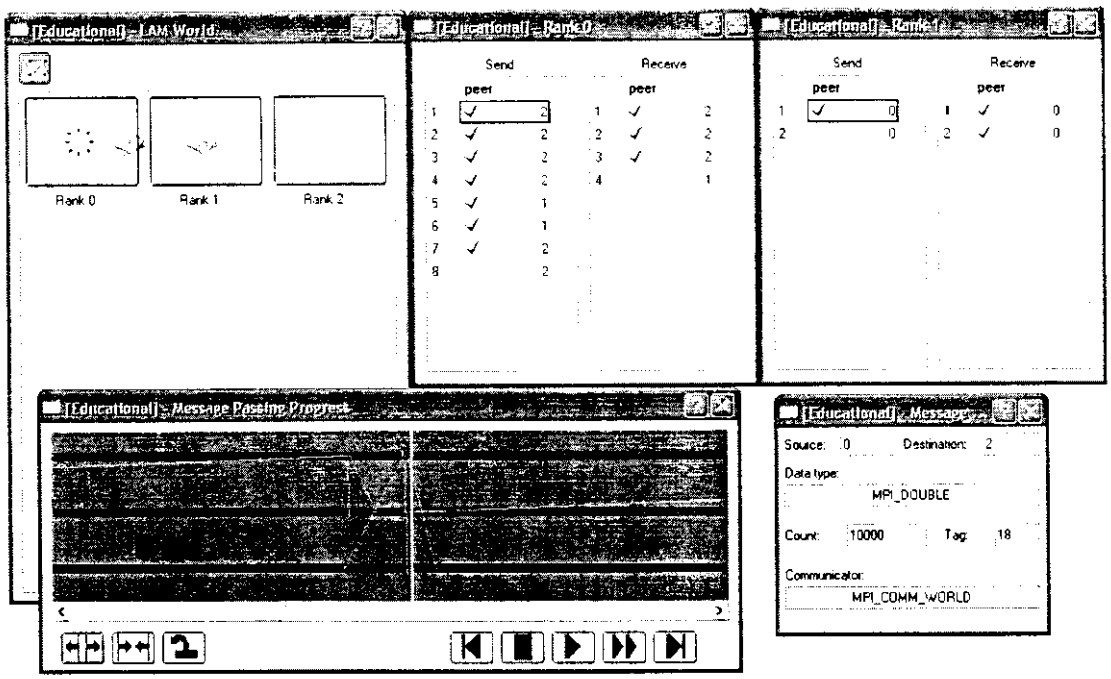
Hình 3-17 Cửa sổ con của BKPD

Người sử dụng cũng có thể thao tác với từng tiến trình thông qua cửa sổ riêng của tiến trình đó. Cửa sổ này cung cấp thêm cho người sử dụng khả năng theo dõi danh sách các break point, theo dõi giá trị stack và các thanh ghi.



Hình 3-18 Cửa sổ giá trị thanh ghi

Người sử dụng có thể theo dõi quá trình thông điệp thông qua các cửa sổ của chương trình BKViewer.



Hình 3-19 Cửa sổ theo dõi truyền thông điệp BKViewer

3.4.3 Định hướng phát triển

Chương trình BKPD cung cấp khá đầy đủ các tiện ích cơ bản hỗ trợ cho việc gỡ rối chương trình song song. Tính tương tác và trực quan giúp người sử dụng có khả năng chẩn đoán lỗi chính xác hơn.

Cả hai thành phần của BKPD – thành phần điều khiển chương trình và thành phần theo dõi truyền thông điệp - đều có những ưu điểm và nhược điểm. Đối với thành phần điều khiển chương trình, ta có các ưu điểm sau:

- BKPD đưa khả năng điều khiển luồng chương trình của gdb, vốn chỉ áp dụng cho các chương trình tuần tự, áp dụng vào các chương trình song song, với một giao diện khá thân thiện.
- Có thể triển khai chương trình trên môi trường Win.
- BKPD được xây dựng theo mô hình client – server. Một server khi được triển khai trên một trung tâm tính toán sẽ có khả năng phục vụ một lúc nhiều người dùng.
- BKPD sẽ được phát triển thành một chương trình mã nguồn mở.

Các nhược điểm của thành phần điều khiển chương trình:

- Chương trình chạy chưa thật sự ổn định, vẫn cần cải tiến trong tương lai.
- Hiện nay trên thế giới chưa có một phần mềm mã nguồn mở nào cung cấp khả năng điều khiển quá trình chạy của chương trình song song giống như BKPD.

Đối với thành phần theo dõi truyền thông điệp, hiện nay trên thế giới có một phần mềm rất phổ biến là Xmpi. Tuy nhiên, Tài liệu này không sử dụng Xmpi mà xây dựng một công cụ riêng BKViewer với mục đích xây dựng các khả năng thích hợp hơn cho BKPD. Một số ưu điểm của BKViewer so với Xmpi là:

- BKViewer cung cấp một giao diện thân thiện hơn Xmpi.
- BKViewer cung cấp khả năng tương tác với quá trình truyền thông tốt hơn Xmpi.
- BKViewer – client có thể triển khai trên môi trường Windows.

Một số nhược điểm của BKViewer so với Xmpi:

- BKViewer vẫn chưa xử lý được toàn bộ các dữ liệu trong trace file như Xmpi.
- Xmpi cho phép theo dõi một số thông số thống kê như tỷ lệ truyền thông/tính toán. BKViewer chưa có khả năng này.
- Xmpi có khả năng portable trên các nền platform khác nhau như Intel, Mac,...

Thực tế, BKPD chưa thể coi là một sản phẩm hoàn thiện. Đây chỉ là nền tảng, là bộ khung để tiếp tục xây dựng một chương trình gỡ rối hoàn thiện.

Một thành quả quý giá không kém so với sản phẩm BKPD, đó là những kiến thức thu được trong quá trình tìm hiểu, nghiên cứu để xây dựng BKPD. Những kiến thức này cũng sẽ là tiền đề để phát triển, cải tiến chương trình trong tương lai.

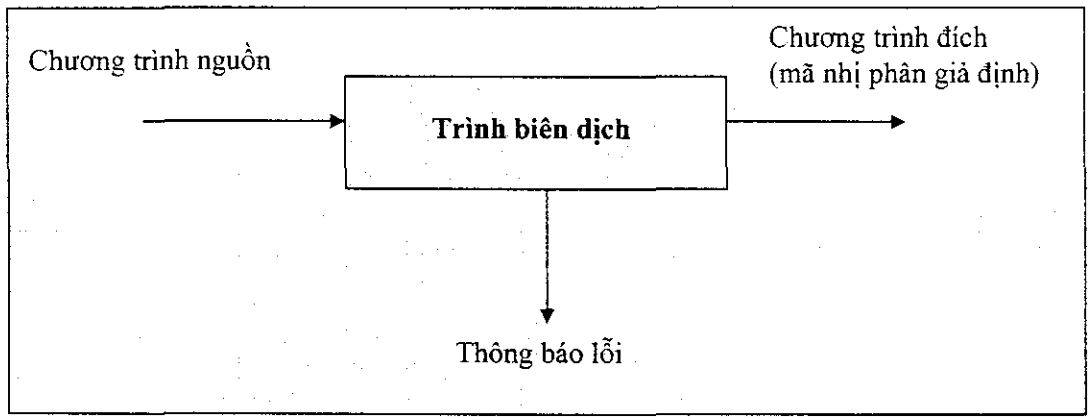
CHƯƠNG 4 Xây Dựng Máy Ảo và Ngôn Ngữ Thông Dịch Đơn Giản

4.1 Tổng quan về chương trình

Chương trình gồm có 2 thành phần:

- **Trình biên dịch :**

Làm nhiệm vụ biên dịch chương trình nguồn là một lớp các bài toán với các phép toán thao tác đơn giản về vector ,ma trận (khởi tạo ma trạn, vector, đọc từ file , in ra file ,v.v....) ra dạng mã nhị phân giả định.



Hình 4-1 Sơ đồ trình biên dịch

+ **Đầu vào:** Chương trình nguồn được viết theo cú pháp giả pascal, ví dụ :

```
//Chương trình nguồn
var
      A : matrix[5,5];
      x,b : vector[5];
      i,j,size : integer;
begin
      readfromfile('matrix',A,'mt3.txt');
      readfromfile('vector',b,'v1.txt');
      exec('vecgetsize',b,size);
      createvector(x,size);
```

```
      exec('sles',A,x,b);

      for i := 0 to size - 1 do write(x[i],'   ');

end.
```

** Các thủ tục, hàm của ngôn ngữ giả pascal xem chi tiết ở phần sau*

+ **Đầu ra:** Chương trình đích (mã nhị phân giả định)

//Chương trình đích

0C 00 44 07 00 00 00 (Lenh JMP, nhảy đến địa chỉ 7)

24 00 03 (Lenh RefreshParaBuffer)

23 00 06 6D 61 74 72 69 78 (Lenh AddStringPara, đưa xâu ký tự 'matrix' ra vùng đệm)

22 00 44 2E 00 00 00 (Lenh AddNumberPara, đưa giá trị 46 tức là địa chỉ của biến A ra vùng đệm)

23 00 07 4D 54 41 2E 54 58 54 (Lenh AddStringPara, đưa xâu ký tự 'MTA.TXT' ra vùng đệm)

11 00 00 (Lenh ReadFromFile, gọi lệnh readfromfile với các tham số lấy từ vùng đệm)

24 00 03 (Lenh RefreshParaBuffer)

23 00 06 76 65 63 74 6F 72 (Lenh AddStringPara, đưa xâu ký tự 'vector' ra vùng đệm)

22 00 44 36 00 00 00 (Lenh AddNumberPara, đưa giá trị 54 tức là địa chỉ của biến b ra vùng đệm)

23 00 0B 56 45 43 54 4F 52 42 2E 54 58 54 (Lenh AddStringPara, đưa xâu ký tự 'VECTORB.TXT' ra vùng đệm)

11 00 00 (Lenh ReadFromFile, thực hiện lệnh readfromfile với các tham số lấy từ vùng đệm)

24 00 03 (Lenh RefreshParaBuffer)

23 00 0A 76 65 63 67 65 74 73 69 7A 65 (Lenh AddStringPara, đưa xâu ký tự 'vecgetsize' ra vùng đệm)

22 00 44 36 00 00 00 (Lenh AddNumberPara, đưa giá trị 54 tức địa chỉ của biến b ra vùng đệm)

22 00 44 3A 00 00 00 (Lenh AddNumberPara, đưa giá trị 58 tức địa chỉ của biến size ra vùng đệm)

50 00 00 (Lenh Exec, thực hiện lời gọi thư viện tính toán với hàn vecgetsize để lấy kích thước của biến vector b đưa vào biến size)

24 00 00 (Lenh RefreshParaBuffer)

22 00 44 32 00 00 00 (Lenh AddNumberPara, đưa giá trị 50 tức địa chỉ của biến x ra vùng đệm)

0F 10 44 3A 00 00 00 (Lenh PUSH, đưa giá trị 58, địa chỉ của biến size vào ngăn xếp)

10 10 44 03 00 00 00 (Lenh POP, lấy giá trị ra khỏi ngăn xếp và đưa vào biến trung gian tại địa chỉ 3)

22 10 44 03 00 00 00 (Lenh AddNumberPara, đưa giá trị của biến trung gian tại địa chỉ 3 ra vùng đệm)

21 00 00 (Lenh CreateVector, thực hiện lệnh createvector với các tham số lấy từ vùng đệm)

24 00 04 (Lenh RefreshParaBuffer)

23 00 04 73 6C 65 73 (Lenh AddStringPara. đưa xâu ký tự 'sles' ra vùng đệm)

22 00 44 2E 00 00 00 (Lenh AddNumberPara. đưa giá trị 46 tức địa chỉ biến A ra vùng đệm)

22 00 44 32 00 00 00 (Lenh AddNumberPara. đưa giá trị 50 tức địa chỉ biến x ra vùng đệm)

22 00 44 36 00 00 00 (Lenh AddNumberPara. đưa giá trị 58 tức địa chỉ biến b ra vùng đệm)

50 00 00 (Lenh Exec, thực hiện gọi thư viện tính toán sles để giải hệ tuyến tính Ax = b)

60 01 14 00 3A 00 00 00 (Lenh Move giá trị 0 vào ô nhớ tại địa chỉ 58 tức biến i)

0F 10 44 3A 00 00 00 (Lenh PUSH cất giá trị biến i vào ngăn xếp)

0F 10 44 42 00 00 00 (Lenh PUSH cất giá trị biến size vào ngăn xếp)

60 01 14 01 12 00 00 00 (Lenh Move giá trị 1 vào biến trung gian tại địa chỉ 18)

10 10 44 03 00 00 00 (Lenh POP lấy giá trị ra khỏi ngăn xếp tức giá trị biến size và đưa vào biến trung gian tại địa chỉ 3)

01 30 44 03 00 00 00 12 00 00 00 21 00 00 00 (Lenh SUB thực hiện trừ nội dung biến tại địa chỉ 3 tức biến size cho biến tại địa chỉ 18 và kết quả đưa vào biến trung gian tại địa chỉ 33)

0F 10 44 21 00 00 00 (Lenh PUSH cất giá trị của biến trung gian tại địa chỉ 33 vào ngăn xếp)

10 10 84 16 00 00 00 (Lenh POP lấy giá trị ra khỏi ngăn xếp, chuyển thành kiểu double và đưa giá trị vào biến trung gian tại địa chỉ 22)

10 10 84 07 00 00 00 (Lenh POP lấy giá trị ra khỏi ngăn xếp từ giá trị biến i và chuyển thành kiểu double và đưa vào biến trung gian tại địa chỉ 7)

09 38 00 07 00 00 00 16 00 00 00 (Lenh Leq thực hiện phép so sánh < nội dung của biến trung gian tại địa chỉ 7 và địa chỉ 22)

0E 00 00 6D 01 00 00 (Lenh JF thực hiện nhảy ra khỏi chương trình nếu cờ bằng 0)

24 00 02 (Lenh RefreshParaBuffer làm tươi vùng đệm truyền tham số)

60 11 44 3A 00 00 00 03 00 00 00 (Lenh Move giá trị ô nhớ có địa chỉ 58 tức biến i vào biến trung gian tại địa chỉ 3)

0F 30 88 32 00 00 00 03 00 00 00 (Lenh PUSH cất giá trị của x[i] vào ngăn xếp)

10 10 88 07 00 00 00 (Lenh POP lấy giá trị của x[i] ra khỏi ngăn xếp và đưa vào biến trung gian tại địa chỉ 7).

22 10 88 07 00 00 00 (Lenh AddNumberPara đưa giá trị của biến trung gian tại địa chỉ 7 ra vùng đệm)

23 00 02 20 20 (Lenh AddStringPara đưa xâu ký tự ' ' ra vùng đệm)

25 00 00 (Lenh WRITE thực hiện ghi các dữ liệu ở vùng đệm ra màn hình)

0F 10 44 3A 00 00 00 (Lenh PUSH cất giá trị biến i vào ngăn xếp)

60 01 14 01 12 00 00 00 (Lenh Move giá trị 1 vào biến trung gian tại địa chỉ 18)

10 10 44 03 00 00 00 (Lenh POP lấy giá trị biến i ra khỏi ngăn xếp và đưa vào biến trung gian tại địa chỉ 3)

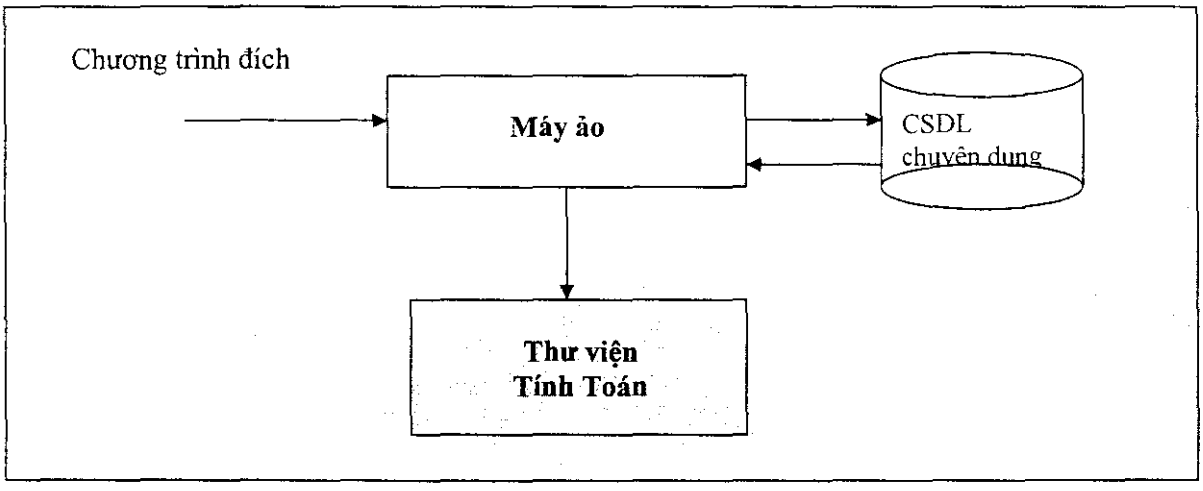
00 30 44 03 00 00 00 12 00 00 00 3A 00 00 00 (Lenh ADD cộng i với 1 và kết quả đưa vào i)

0C 00 00 BF 00 00 00 (Lenh JMP nhảy lên nhãn của vòng lặp)

** Tham khảo chi tiết mã lệnh của máy ảo ở phần sau*

Máy ảo:

Thực chất chính là 1 chương trình LAM/MPI được xây dựng dựa trên hệ thư viện tính toán khoa học PETSC, nhận đối số đầu vào là các mã lệnh trong file Chương trình đích (mã nhị phân giả định), từ đó thực hiện các thao tác tính toán cho ra kết quả cuối cùng .



Hình 4-2 Máy ảo

+ **Đầu vào**: Chương trình đích ở trên

+ **Đầu ra** :

```
Kết quả thực hiện chương trình

Trong chương trình trên dữ liệu cho ma trận A được đọc từ file
mt3.txt có nội dung như sau:

5 5

1 4 1 2 1

2 1 2 3 2

0 0 1 3 0

0 0 0 2 1

1 1 1 1 1

Dữ liệu vector b được đọc từ file v1.txt có nội dung như sau:

5
```

```
2 1 1 2 0


Màn hình hiển thị kết quả:

1.750000   0.250000   -2.750000   1.250000   -0.500000
```

Kết quả thực hiện chương trình

Trong chương trình trên dữ liệu cho ma trận A được đọc từ file mt3.txt có nội dung như sau:
5 5
1 4 1 2 1
2 1 2 3 2
0 0 1 3 0
0 0 0 2 1
1 1 1 1 1

Dữ liệu vector b được đọc từ file v1.txt có nội dung như sau:
5
2 1 1 2 0

4.2 Xây dựng ngôn ngữ lập trình

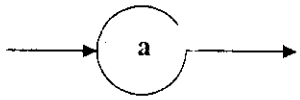
Cú pháp ngôn ngữ lập trình có thể được biểu diễn dưới dạng BNF hoặc dưới dạng sơ đồ cú pháp.

Trong phần này, cú pháp của ngôn ngữ lập trình được biểu diễn dưới dạng sơ đồ cú pháp với mục đích áp dụng phương pháp phân tích đệ quy trên dưới.

Xét một văn phạm G = (N, T, S, P) là văn phạm phi ngữ cảnh, mọi quy tắc A $\rightarrow \alpha 1 \mid \alpha 2 \mid \ldots \mid \alpha k$ dạng BNF được chuyển thành sơ đồ cú pháp theo các qui định sau:

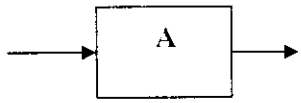
1- Nếu A $\rightarrow \alpha 1 \mid \alpha 2 \mid \ldots \mid \alpha k$ là quy tắc của G được ánh xạ thành biểu đồ đoán nhận A, có cấu trúc xác định bởi vế phải các sản xuất theo qui định từ 2÷ 6.

2- Mỗi $a \in T$ lập sơ đồ cú pháp:

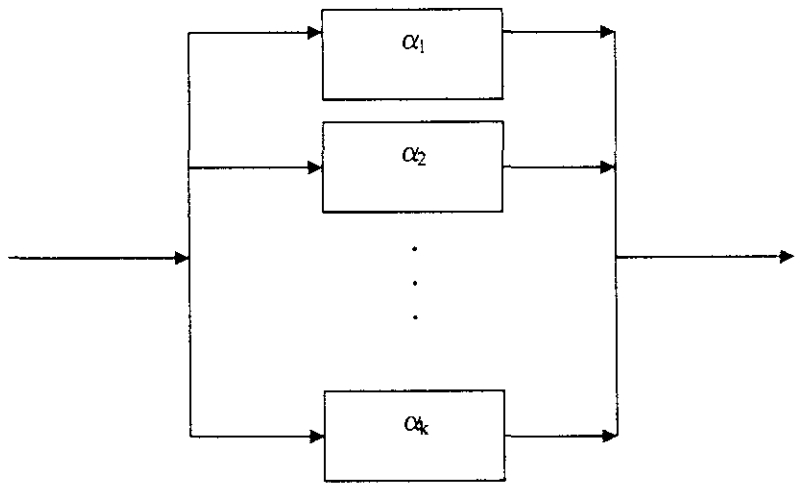


Để tiện cho việc trình bầy trong khuôn khổ trang giấy, hình oval ◯ có thể được dùng thay thế cho hình tròn biểu diễn cho các từ khoá, tên chuẩn, kiểu dữ liệu.

3- Mỗi A ∈ N lập sơ đồ cú pháp:

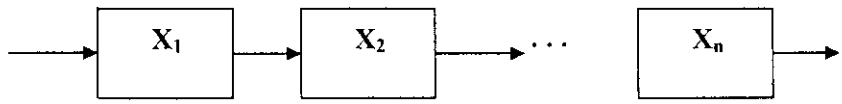


4- Quy tắc A -> α1 | α2 | α3 | . . .|αk lập sơ đồ cú pháp:



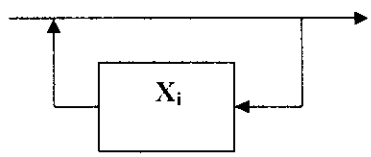
với αi (1 ≤i ≤k) lập sơ đồ cú pháp bằng cách áp dụng quy định 2÷6 .

5- Giả sử αi có dạng X1X2X3. . .Xn, lập sơ đồ cú pháp :



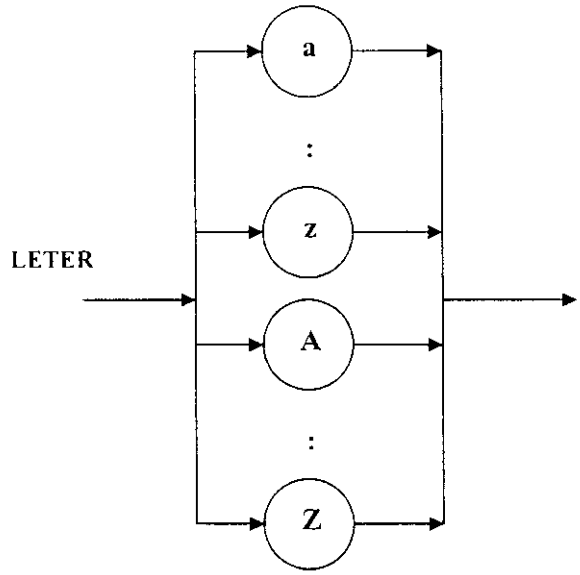
với Xi (1 ≤i ≤n) lập sơ đồ cú pháp bằng cách áp dụng quy định 2÷6.

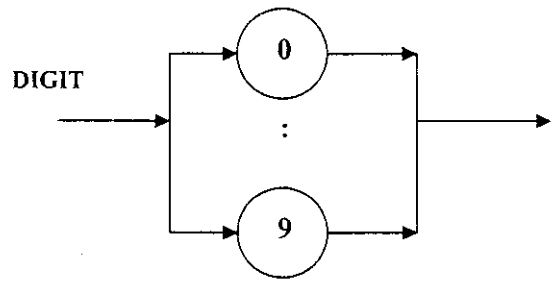
6- Giả sử αi có dạng lặp ký hiệu αi = {Xi}, lập sơ đồ cú pháp:

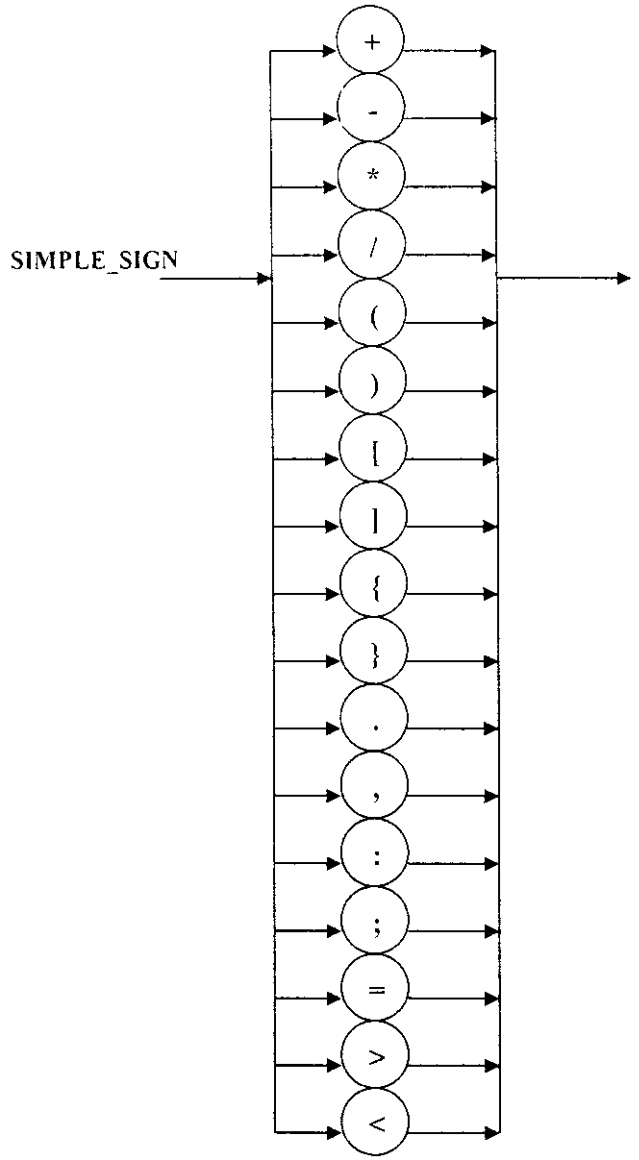


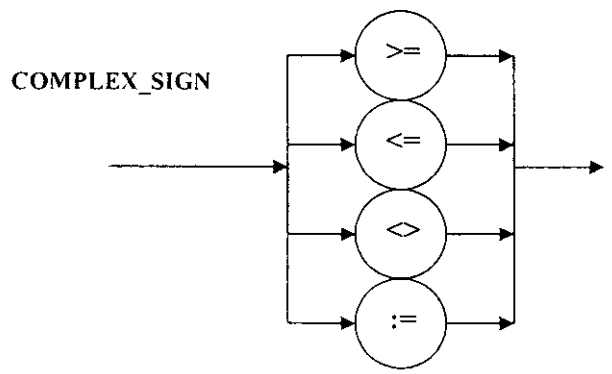
với Xi lập sơ đồ cú pháp bằng cách áp dụng quy định 2÷6.

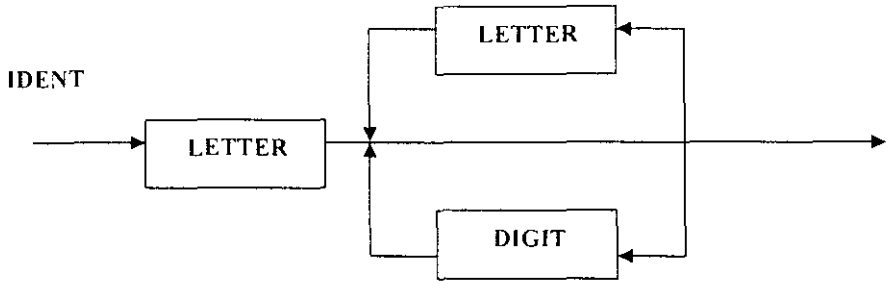
Áp dụng các biểu diễn trên vào cú pháp của ngôn ngữ lập trình được xây dựng.

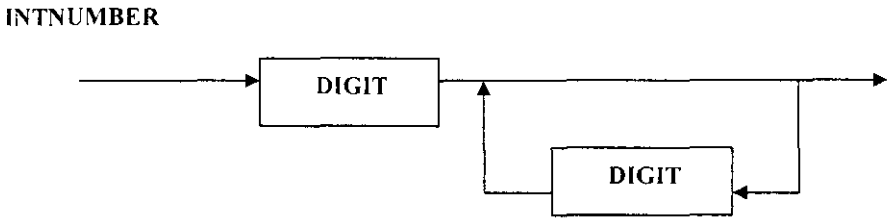


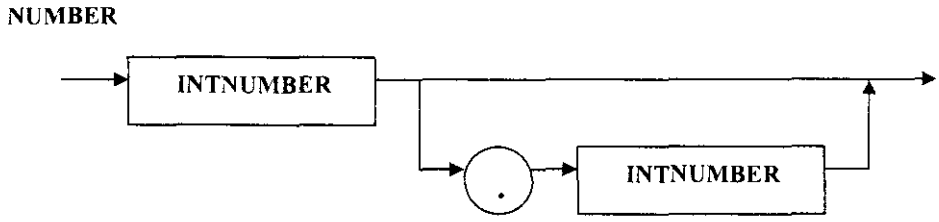


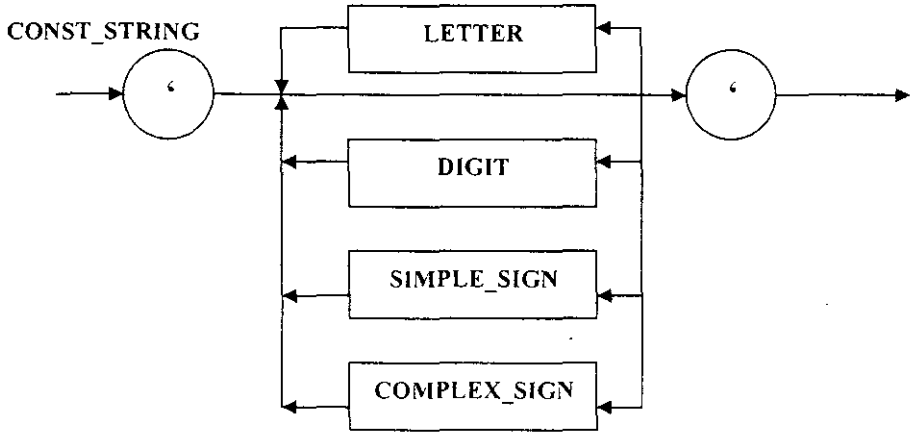

SIMPLE_SIGN
+
-
*
/
(
)
[
]
{
}
.
,
:
;
=
>
<

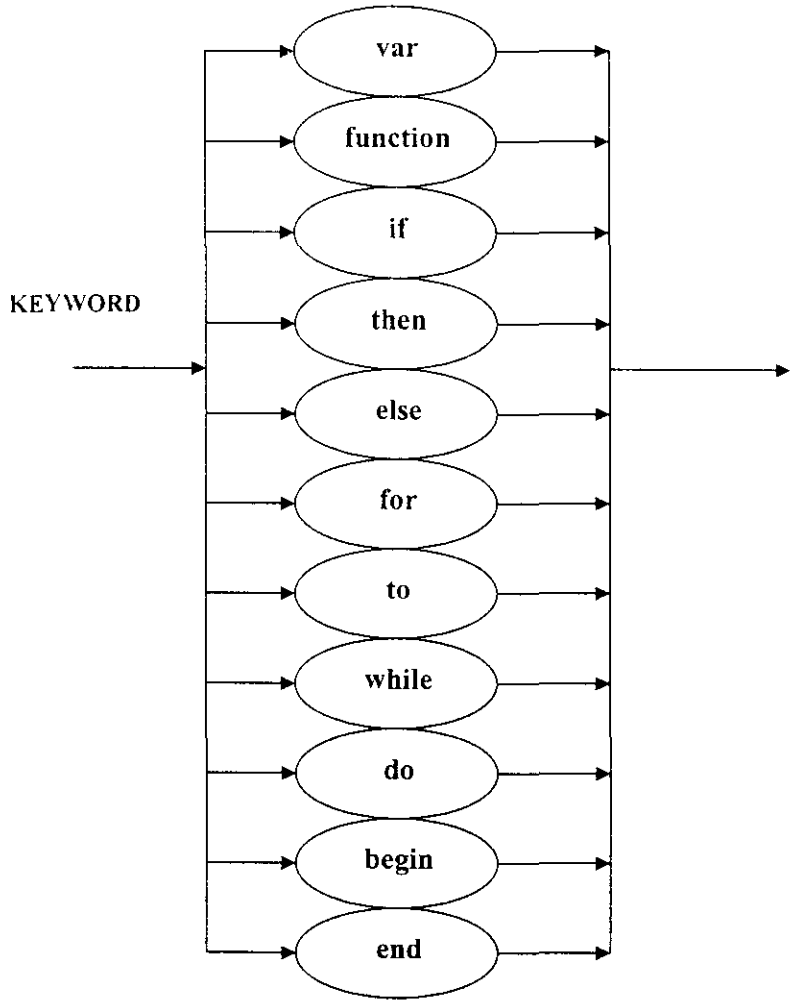

COMPLEX_SIGN
>=
<=
<>
:=

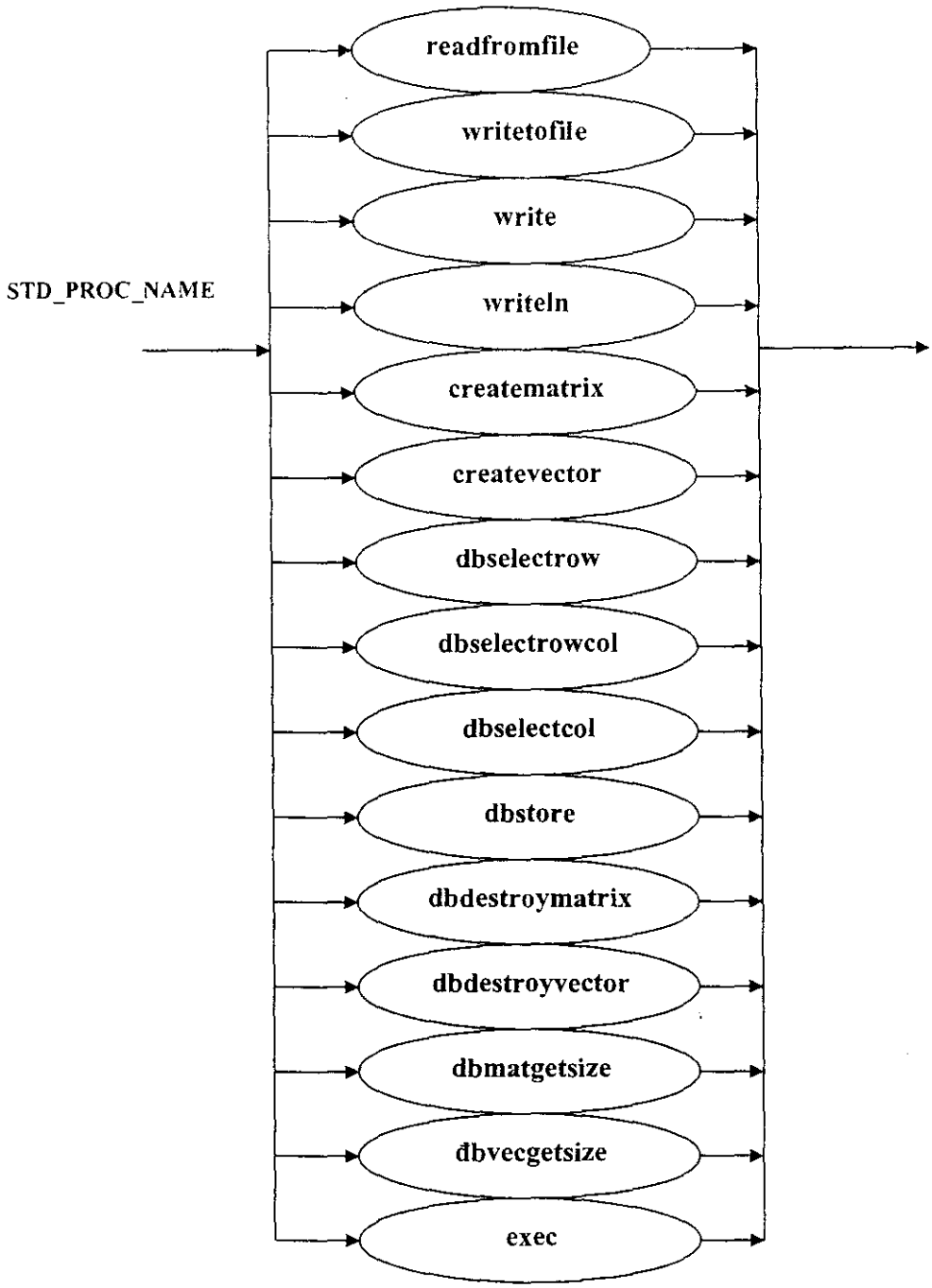

IDENT
LETTER
LETTER
DIGIT

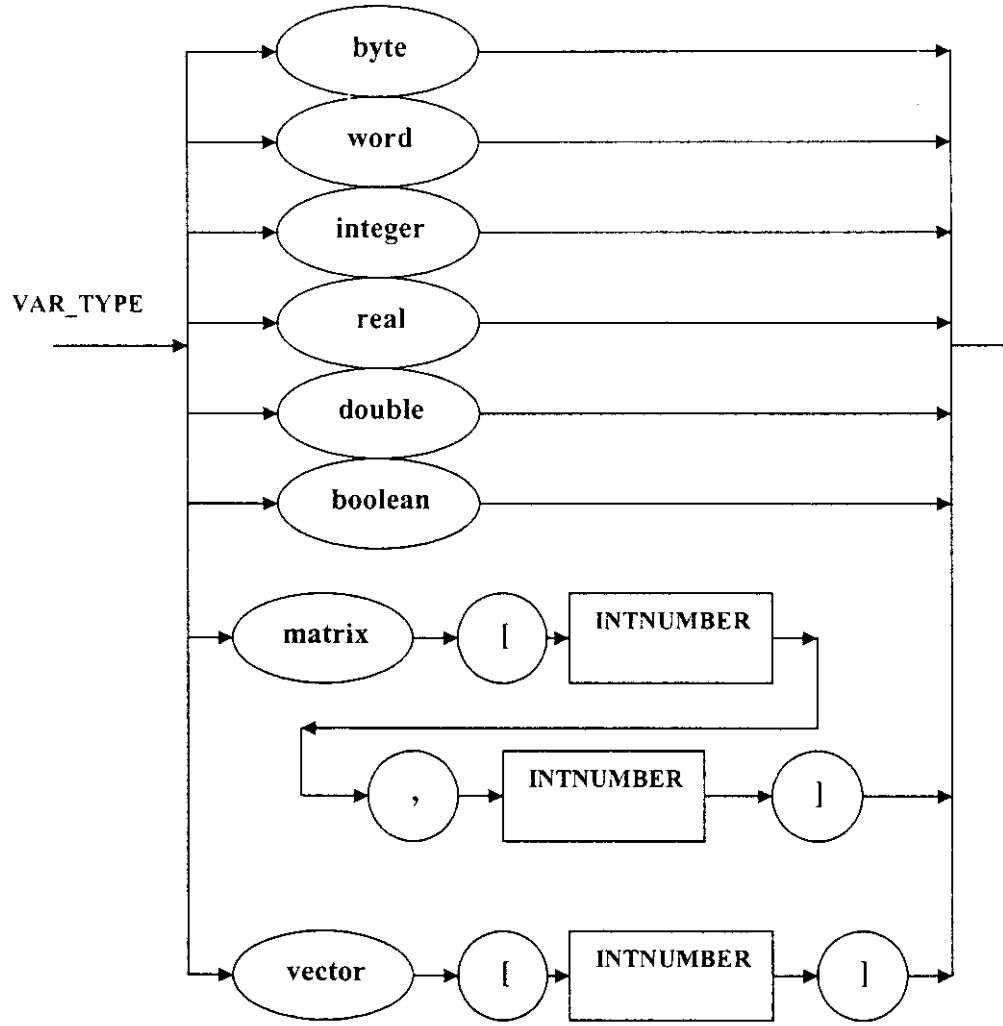

INTNUMBER
DIGIT
DIGIT

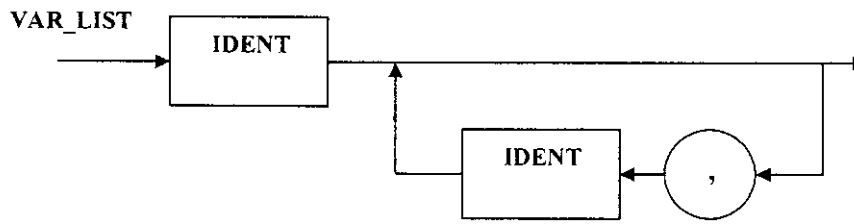

NUMBER
INTNUMBER
.
INTNUMBER

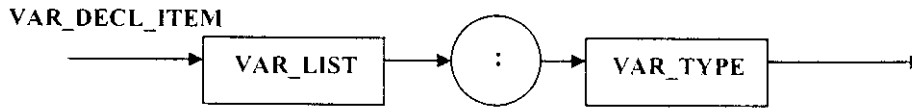

CONST_STRING
'
LETTER
'
DIGIT
SIMPLE_SIGN
COMPLEX_SIGN


KEYWORD
var
function
if
then
else
for
to
while
do
begin
end

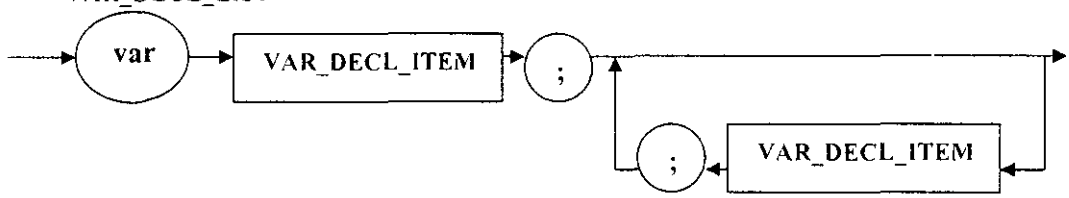

STD_PROC_NAME
readfromfile
writetofile
write
writeln
creatematrix
createvector
dbselectrow
dbselectrowcol
dbselectcol
dbstore
dbdestroymatrix
dbdestroyvector
dbmatgetsize
dbvecgetsize
exec


VAR_TYPE
byte
word
integer
real
double
boolean
matrix
[
INTNUMBER
,
INTNUMBER
]
vector
[
INTNUMBER
]

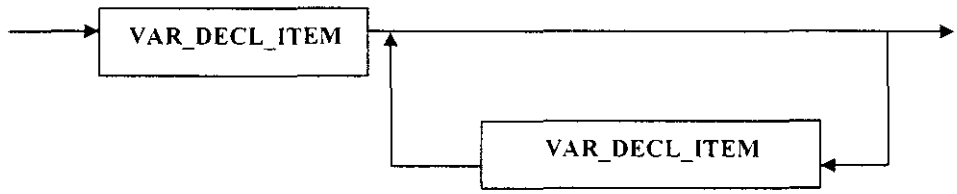

VAR_LIST
IDENT
IDENT
,


VAR_DECL_ITEM
VAR_LIST
:
VAR_TYPE

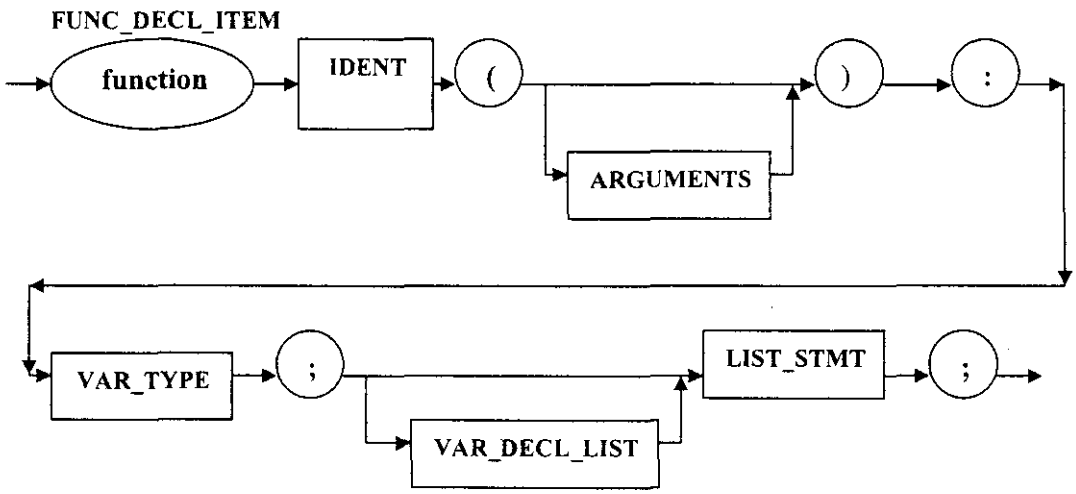
VAR_DECL_LIST



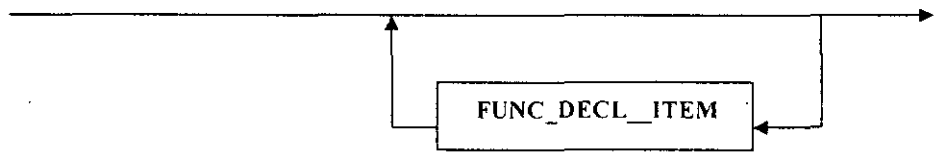
ARGUMENTS



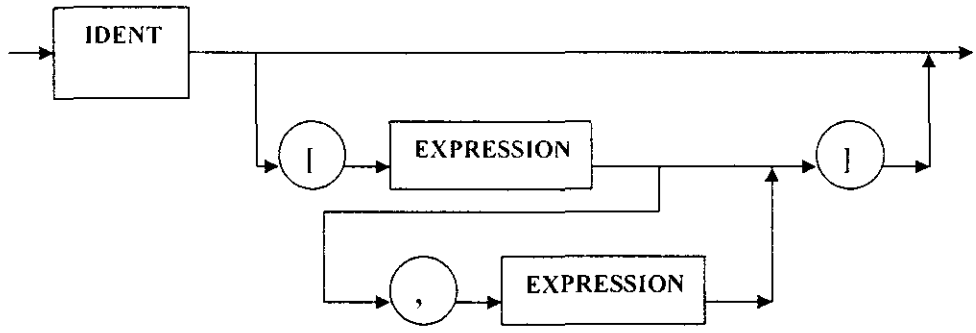
FUNC_DECL_ITEM



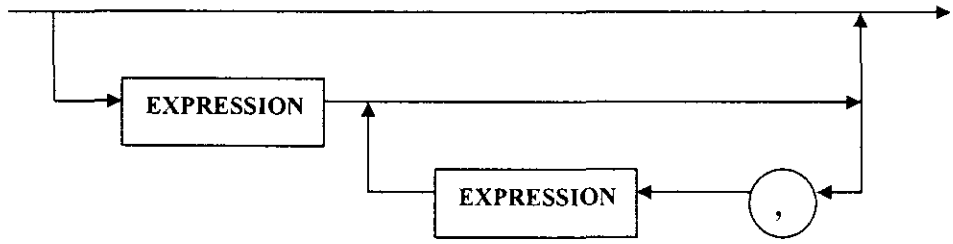
FUNC_DECL_LIST



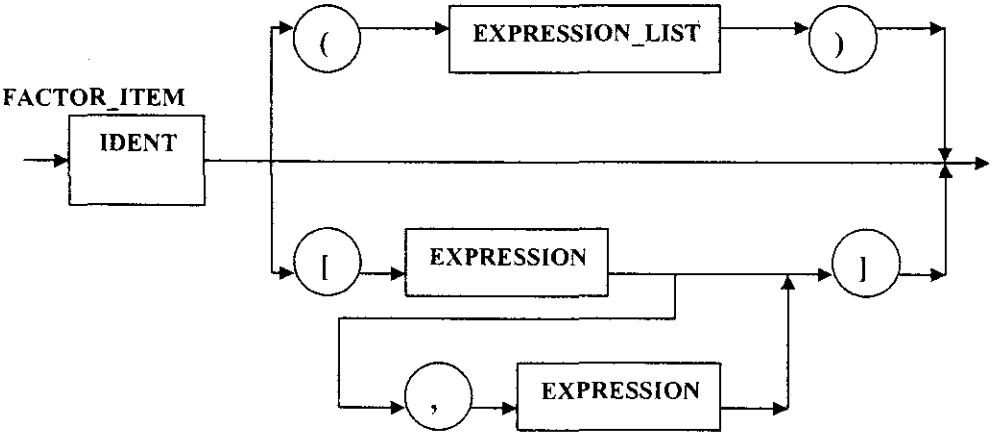
VARIABLE

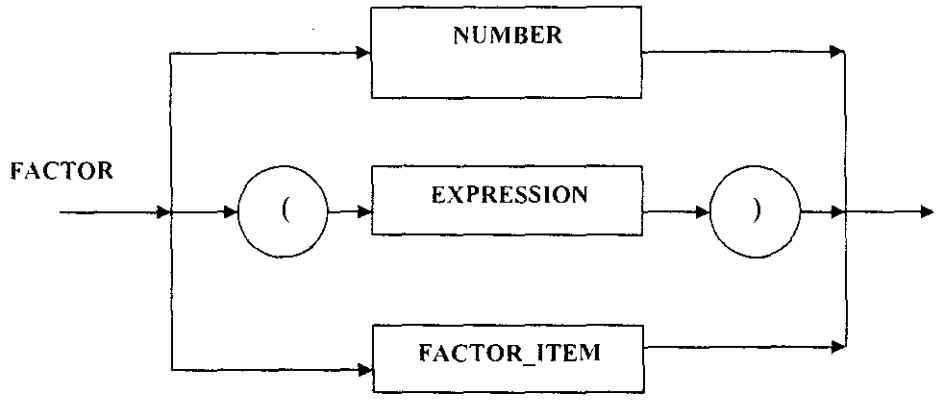


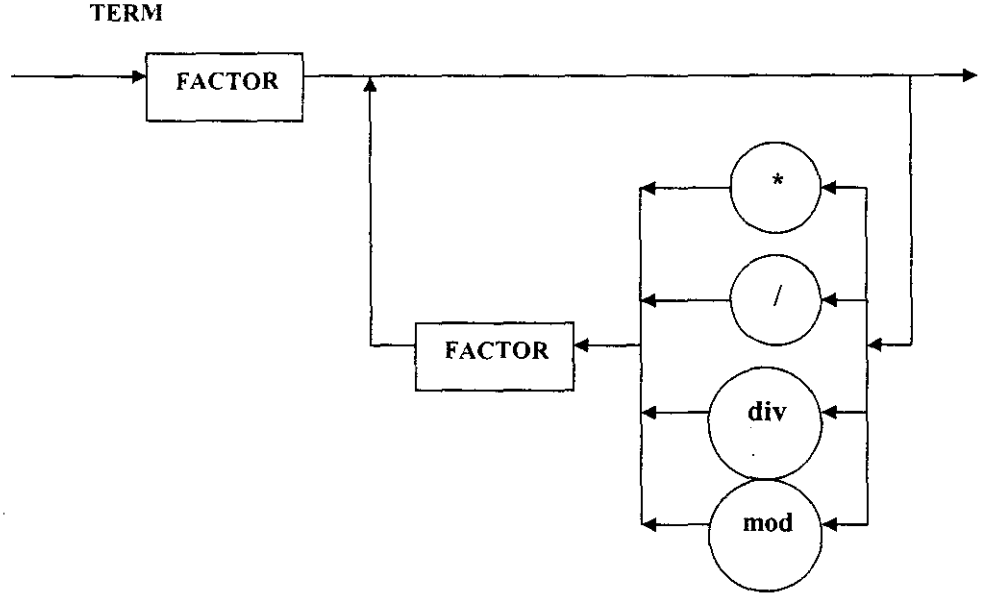
EXPRESSION_LIST

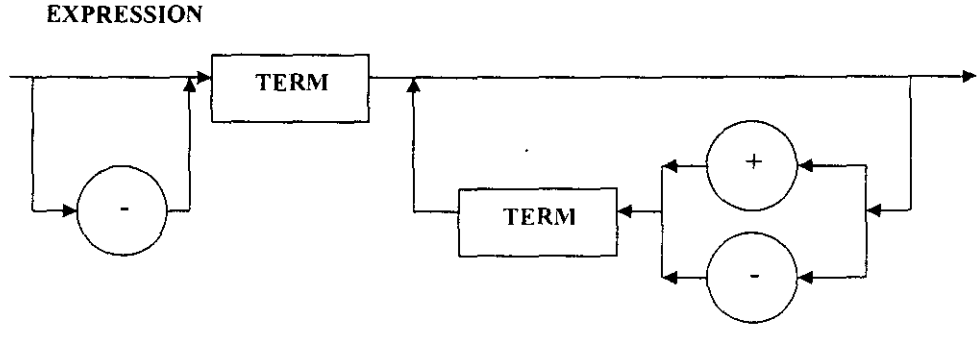


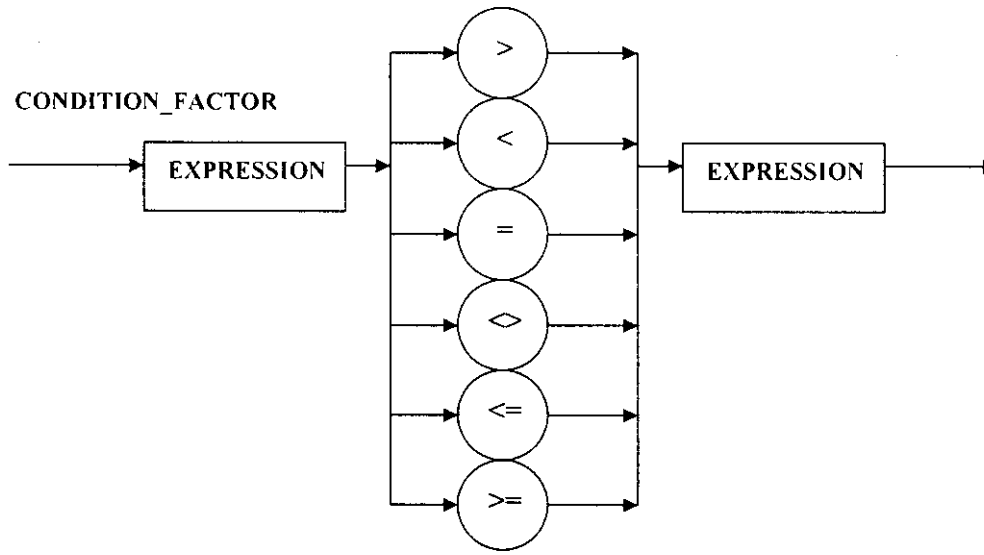
FACTOR_ITEM

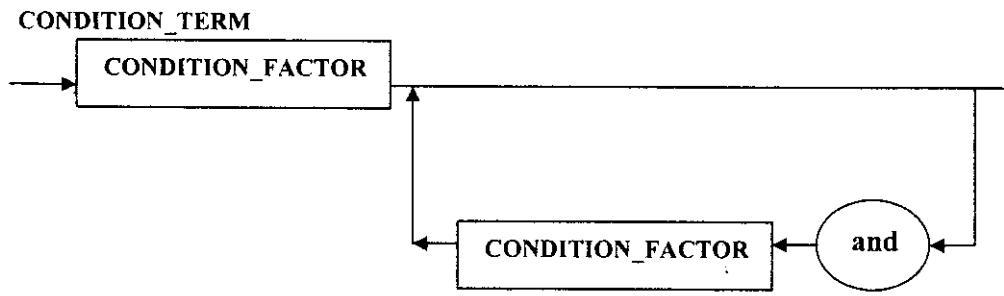


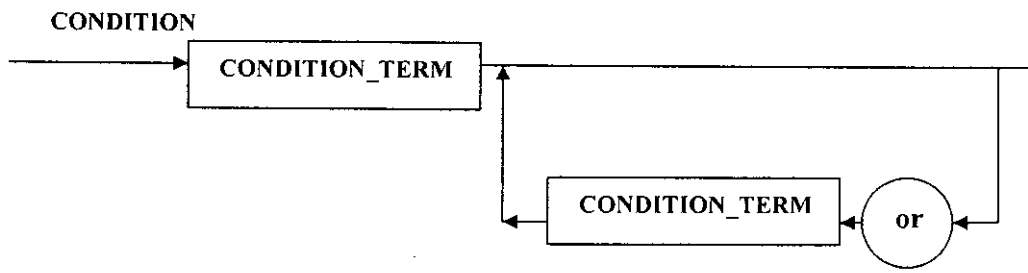

FACTOR
NUMBER
(
EXPRESSION
)
FACTOR_ITEM


TERM
FACTOR
*
/
div
mod
FACTOR

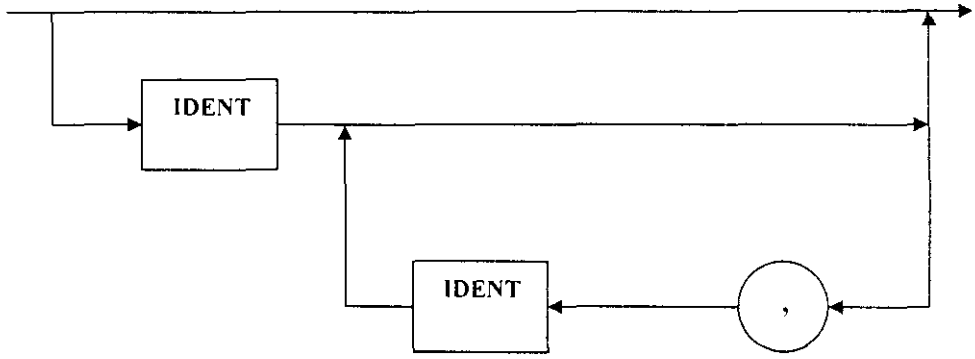

EXPRESSION
TERM
-
TERM
+
-


CONDITION_FACTOR
EXPRESSION
>
<
=
<>
<=
>=
EXPRESSION

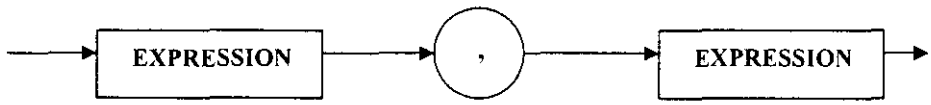

CONDITION_TERM
CONDITION_FACTOR
CONDITION_FACTOR
and


CONDITION
CONDITION_TERM
CONDITION_TERM
or

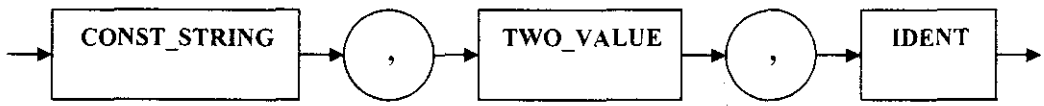
EXEC_PARA_LIST

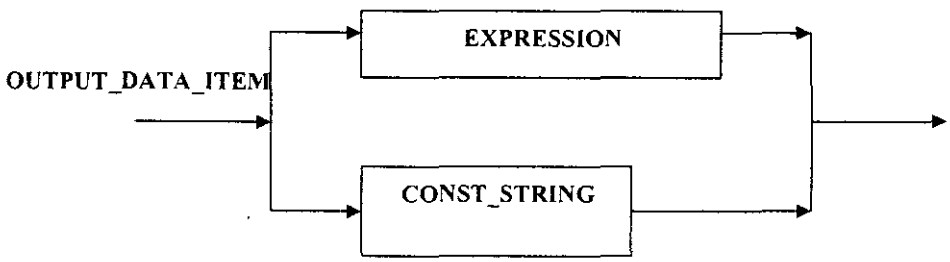


TWO_VALUE

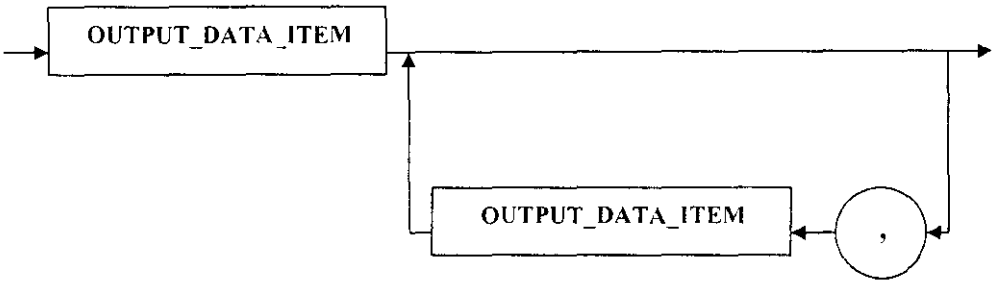


SELECT_PARA

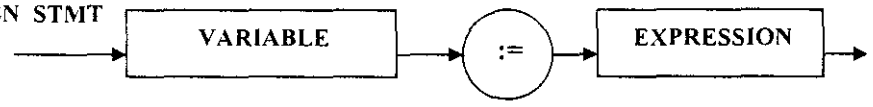




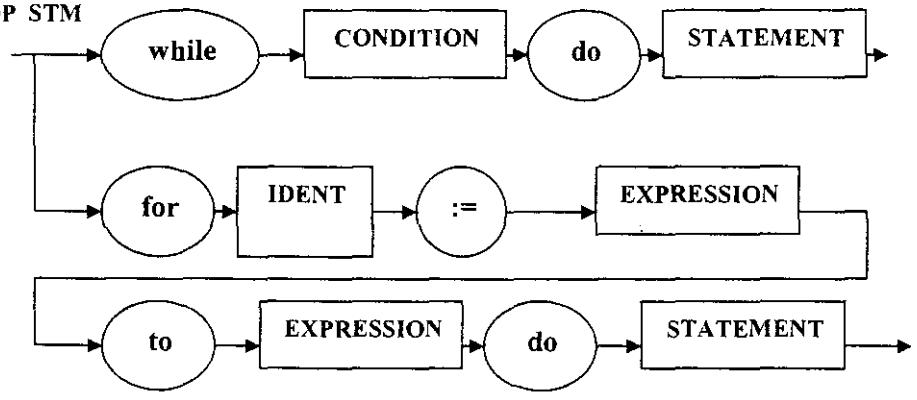
OUTPUT_DATA_LIST



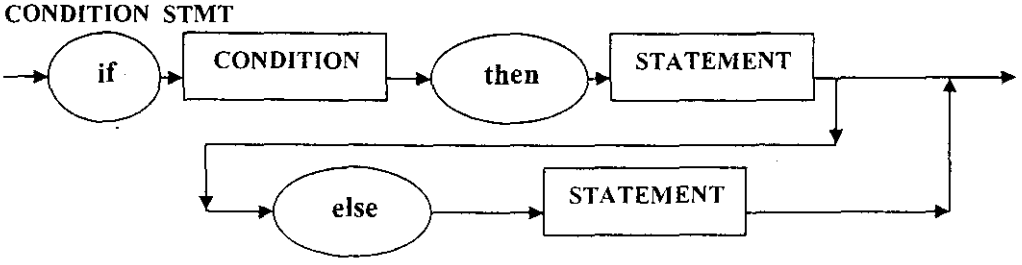
ASSIGN STMT



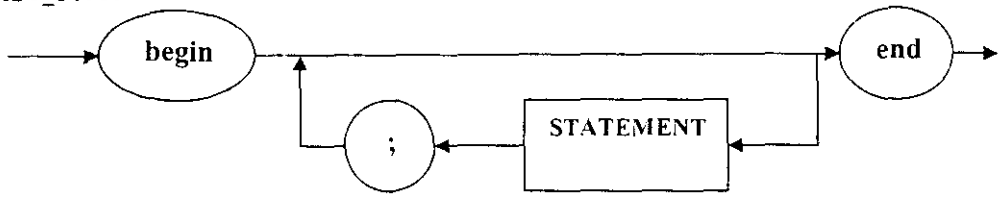
LOOP STM



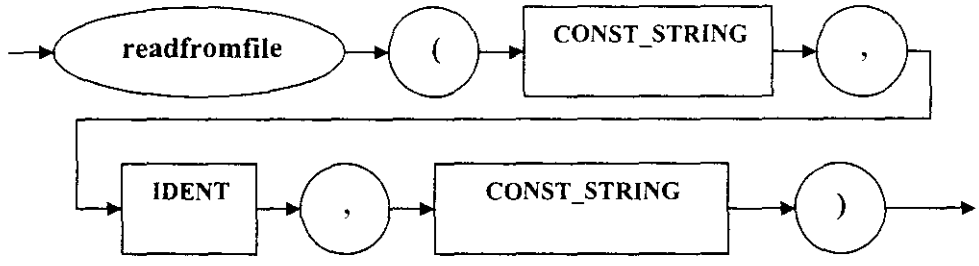
CONDITION STMT



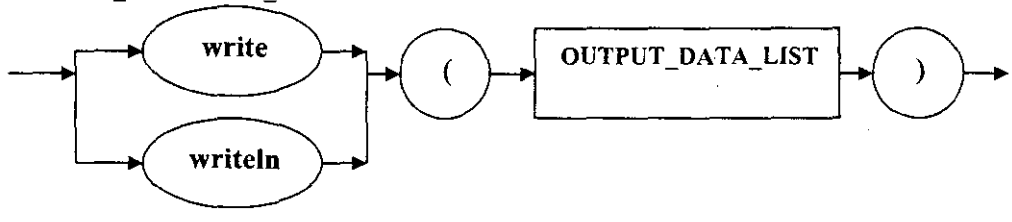
LIST_STMT



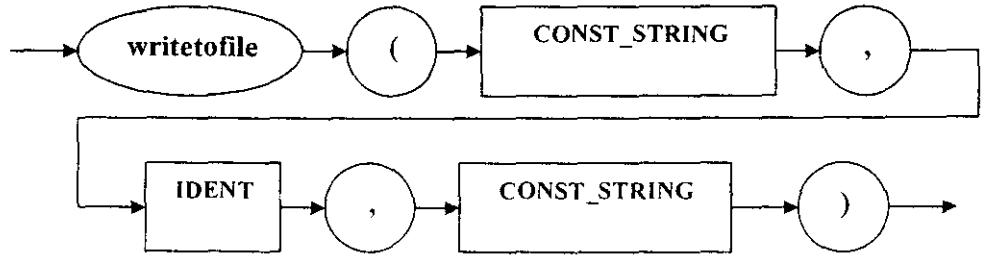
INPUT_STMT

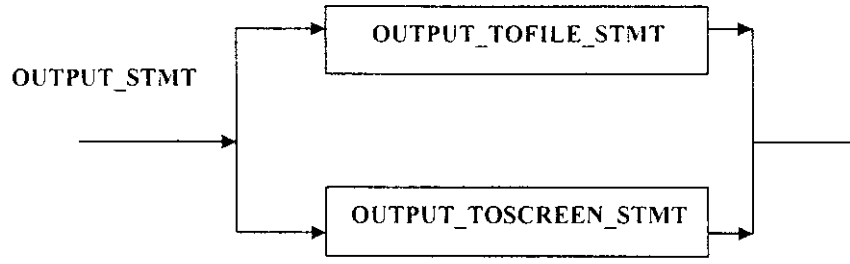


OUTPUT_TOSCREEN_STMT

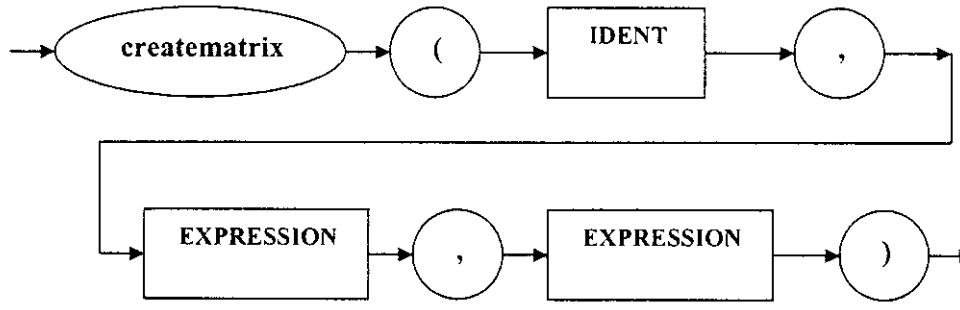


OUTPUT_TOFILE_STMT

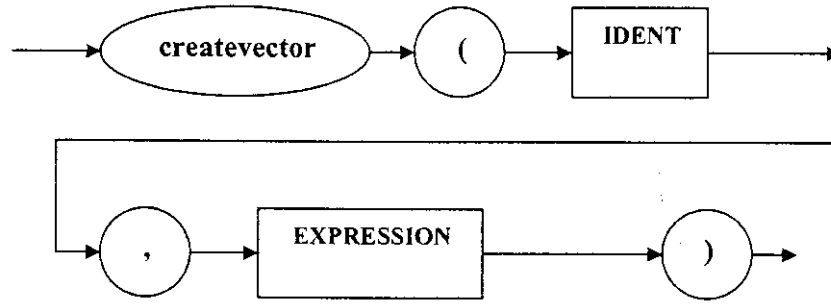




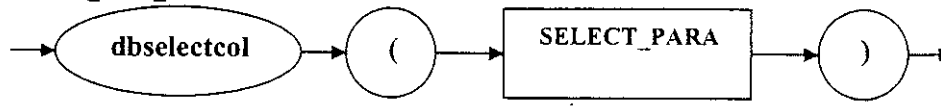
CREATE_MATRIX_STMT



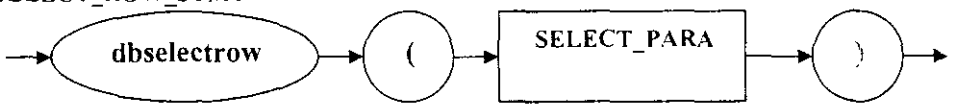
CREATE_VECTOR_STMT



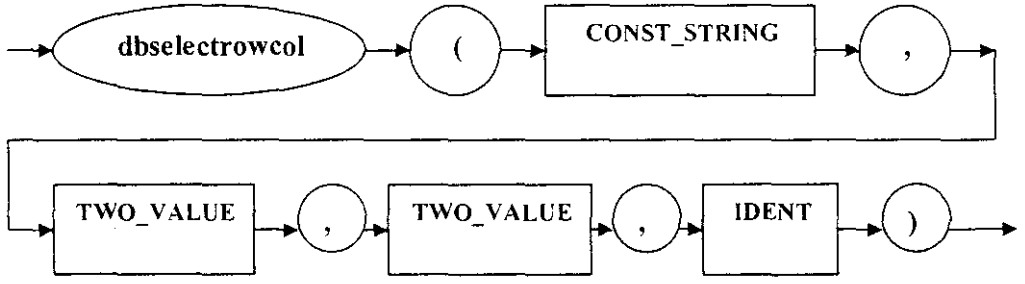
SELECT_COL_STMT



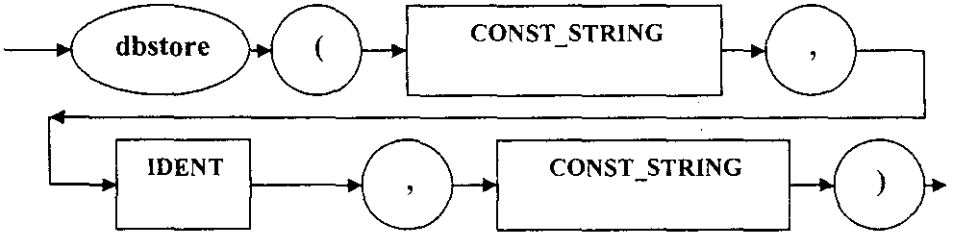
SELECT_ROW_STMT



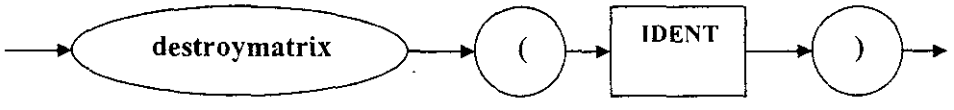
SELECT_ROWCOL_STMT



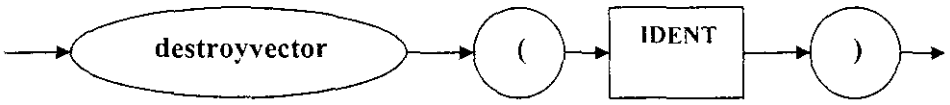
STORE_STMT



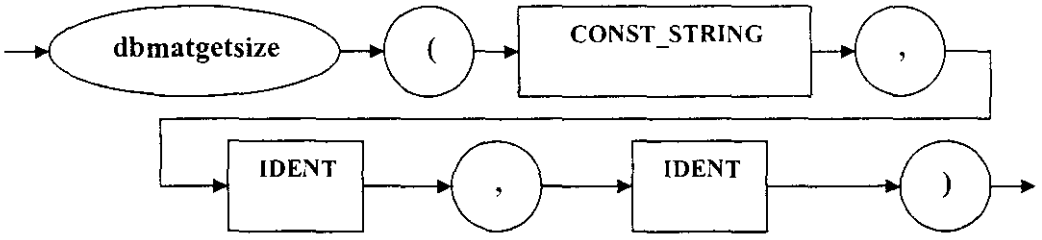
DESTROY_MATIX_STMT



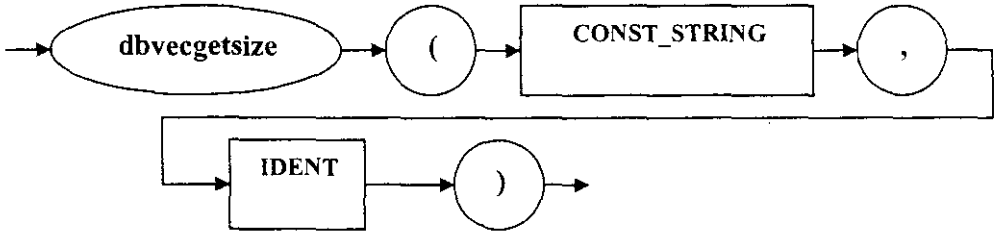
DESTROY_VECTOR_STMT



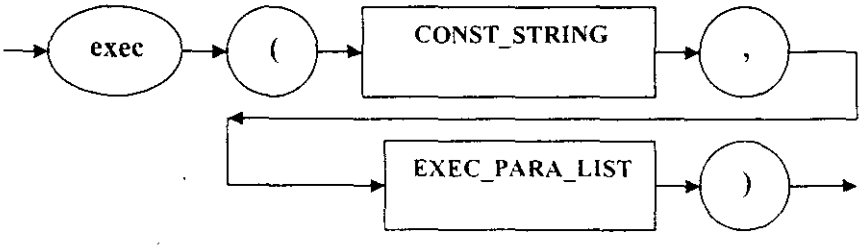
MAT_GET_SIZE_STMT

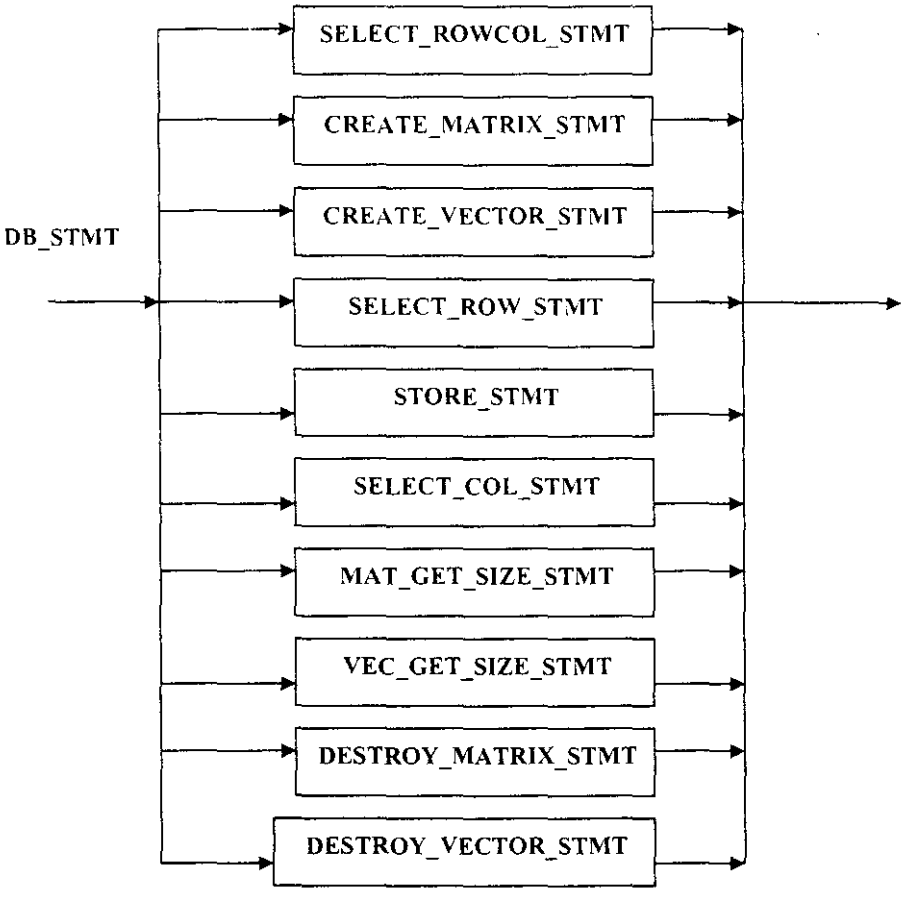


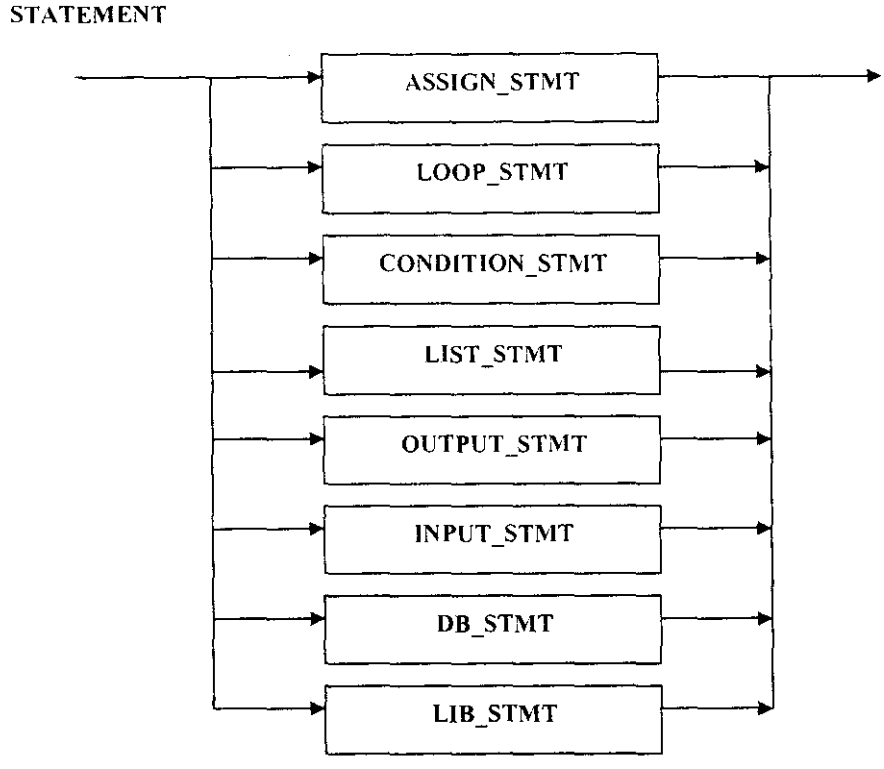
VEC_GET_SIZE_STMT

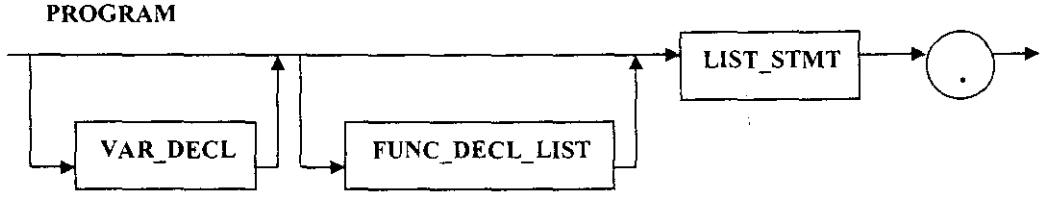


LIB_STMT




DB_STMT
SELECT_ROWCOL_STMT
CREATE_MATRIX_STMT
CREATE_VECTOR_STMT
SELECT_ROW_STMT
STORE_STMT
SELECT_COL_STMT
MAT_GET_SIZE_STMT
VEC_GET_SIZE_STMT
DESTROY_MATRIX_STMT
DESTROY_VECTOR_STMT





Chú thích về chức năng của các thủ tục chuẩn trong ngôn ngữ lập trình nêu trên:

- Thủ tục write, writeln dùng để hiển thị dữ liệu ra màn hình.
- Thủ tục readfromfile dùng để đọc dữ liệu là ma trận hoặc vector số từ file văn bản.
- Thủ tục writetofile dùng để ghi dữ liệu ma trận hoặc vector ra file văn bản.
- Thủ tục creatematrix dùng để khởi tạo dữ liệu cho ma trận trên bộ nhớ.
- Thủ tục createvector dùng để khởi tạo dữ liệu cho vector trên bộ nhớ.
- Thủ tục dbselectrow dùng để lấy dữ liệu ma trận từ cơ sở dữ liệu (chọn một số hàng của ma trận).

- Thủ tục dbselectcol dùng để lấy dữ liệu ma trận từ cơ sở dữ liệu (chọn một số cột của ma trận).
- Thủ tục dbselectrowcol dùng để lấy dữ liệu ma trận từ cơ sở dữ liệu (chọn ra một số hàng và một số cột).
- Thủ tục dbstore dùng để lưu dữ liệu ma trận, vector từ bộ nhớ vào cơ sở dữ liệu.
- Thủ tục dbmatgetsize dùng để lấy kích thước của ma trận trong cơ sở dữ liệu.
- Thủ tục dbvecgetsize dùng để lấy kích thước của vector trong cơ sở dữ liệu.
- Thủ tục dbdestroymatrix dùng để giải phóng dữ liệu ma trận đã được cấp phát trên bộ nhớ.
- Thủ tục dbdestroyvector dùng để giải phóng dữ liệu vector đã được cấp phát trên bộ nhớ.
- Thủ tục exec dùng để gọi thư viện tính toán.

Đối với hai thủ tục readfromfile và writetofile dùng để đọc, ghi dữ liệu ma trận, vector đối với file. Các file này là các file văn bản có định dạng như sau:

Trường hợp file lưu ma trận : Dòng đầu ghi 2 số nguyên M, N xác định kích thước hàng, cột của ma trận. Trong M dòng tiếp theo, mỗi dòng ghi N số thực, các số cách nhau ít nhất 1 dấu cách.

Trường hợp file lưu vector : Dòng đầu ghi số nguyên M xác định kích thước của vector, dòng thứ hai ghi M số thực(các thành phần của vector) cách nhau bởi dấu cách.

Kiểu dữ liệu matrix, vector là mảng 2 chiều và mảng 1 chiều, mỗi phần tử là một số có kiểu double.

Ví dụ 2.1

Ví dụ đoạn chương trình tính ước số chung lớn nhất của hai số nguyên như sau:

```
var a,b :  integer;
function USCLN(a,b : integer) : integer;
begin
     while a <> b do
          if a>b then a := a-b else b := b-a;
```

```
            USCLN :=a;
      end;

      {chuong trinh chinh}
      begin
            write(' uscln cua 32 và 80 là : ', USCLN(32,80));
      end.
```

Ví dụ 2.2.

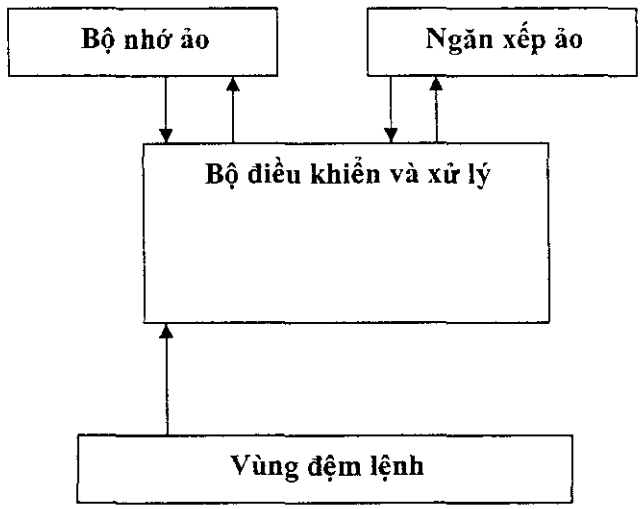
Đoạn chương trình sau đọc dữ liệu ma trận mt, các vector x, y từ file văn bản, sau đó gọi thủ tục trong thư viện tính toán để thực hiện phép tính w = mt*x+y và ghi dữ liệu của vector w ra file như sau:

```
var
mt : matrix[100,100];
x,y,w : vector[100];
begin
      readfromfile ( 'matrix', mt , 'matrix1.txt');
      {doc du lieu ma tran  tu file matrix1.txt}
      readfromfile ('vector', x , 'vectorx.txt');
      {doc du lieu  vector  tu file vectorx.txt}
      readfromfile ('vector', y , 'vectory.txt');
      {doc du lieu  vector  tu file vectory.txt}

      exec ('matmultadd', mt, x, y, w ); { w = mt * x + y}
      writetofile ('vector', w, 'vectorw.txt');
      {ghi du lieu vector ra file vectorw.txt}
end.
```

4.3 Thiết kế máy ảo

4.3.1 Kiến trúc máy ảo

Máy ảo có chức năng đọc và thực thi chương trình mã giả định.
Máy ảo có bộ nhớ, ngăn xếp, vùng đệm lệnh, con trỏ lệnh,biến cờ phục vụ cho việc lưu trữ dữ liệu và thực hiện tính toán.
Sơ đồ thực thi của máy ảo như sau:



Hình 4-3 Sơ đồ thực thi của máy ảo.

4.3.1.1 Bộ nhớ.

Bộ nhớ của máy ảo được tổ chức một cách tuyến tính, kích thước tối đa là 65536 Byte.

Bộ nhớ này gồm 65536 byte được đánh số thứ tự từ 0 đến 65535, dùng để chứa các biến được khai báo trong chương trình. Bộ nhớ này được gọi là bộ nhớ ảo. Ô nhớ gồm k byte liên tiếp nhau (k = 1,2,4,8) bắt đầu từ vị trí j được goi là ô nhớ có địa chỉ j (địa chỉ này được gọi là địa chỉ ảo vì nó không phải là địa chỉ thực sự của ô nhớ đó trong bộ nhớ của máy tính). Trong các phần tiếp theo, nếu không chú thích gì thêm thì khái niệm địa chỉ được hiểu là địa chỉ ảo trong bộ nhớ ảo.

Phần đầu (45 byte) chứa các biến tạm (lưu các giá trị trung gian, thường được dùng trong việc tính toán giá trị của biểu thức) được tổ chức như sau :

- 3 ô nhớ 1 byte được định vị tại các địa chỉ 0 , 15 , 30
- 3 ô nhớ 2 byte được định vị tại các địa chỉ 1 , 16 , 31
- 3 ô nhớ 4 byte được định vị tại các địa chỉ 3 , 18 , 33
- 3 ô nhớ 8 byte được định vị tại các địa chỉ 7 , 22 , 37
- Biến cờ được qui định là ô nhớ 1 byte tại địa chỉ 45.

Phần tiếp theo bắt đầu từ địa chỉ 46 trở đi được định vị cho các biến được khai báo trong chương trình nguồn, các biến này được định vị một cách liến tiếp nhau, biến khai báo trước thì được định vị trước.Địa chỉ của một biến phụ thuộc vào địa chỉ và kích thước của biến trước đó.

4.3.1.2 Ngăn xếp

Ngăn xếp dùng để chứa các kết quả trung gian trong quá trình tính giá trị của biểu thức. Ngăn xếp cũng dùng để cất giá trị của các biến , địa chỉ của lệnh trong các lời gọi chương trình con.

Ngăn xếp được tổ chức một cách tuyến tính với kích thước tối đa của ngăn xếp là 65536 byte.

Với ngăn xếp có hai thao tác cơ bản là cất dữ liệu vào ngăn xếp và lấy dữ liệu ra khỏi ngăn xếp. Hai thao tác này có các thông tin điều khiển đi cùng để xác định giá trị hoặc địa chỉ của dữ liệu và kích thước của dữ liệu.Kích thước dữ liệu có thể là 1 , 2 , 4 hoặc 8 byte.

Ngăn xếp được quản lý bởi một con trỏ trỏ đến đỉnh của ngăn xếp.Ban đầu ngăn xếp không có dữ liệu, con trỏ ngăn xếp có giá trị bằng 0.

4.3.1.3 Con trỏ lệnh

Con trỏ lệnh là một giá trị nguyên 4 byte xác định địa chỉ của lệnh sẽ được thực thi. Sau mỗi lệnh, con trỏ lệnh sẽ được tự động cập nhật để trỏ sang lệnh tiếp theo.

4.3.1.4 Biến cờ

Biến cờ có kích thước 1 byte tại địa chỉ 45 trong bộ nhớ máy ảo.
Biến cờ dùng để chứa kết quả của một phép so sánh. Nếu kết quả của phép so sánh là đúng thì giá trị của thanh ghi cờ được thiết lập bằng 1 và nếu kết quả của một phép so sánh là sai thì thanh ghi cờ được thiết lập giá trị bằng 0. Giá trị của biến cờ có tác động đến các lệnh nhảy.

4.3.1.5 Vùng đệm truyền tham số

Vùng đệm truyền tham số có kích thước tối đa là 65536 được tổ chức một cách tuyến tính.

Vùng đệm này dùng để chứa giá trị của các tham số trong các câu lệnh :

- Đọc ghi dữ liệu đối với file.
- Hiển thị dữ liệu ra màn hình.
- Câu lệnh gọi thư viện tính toán.
- Câu lệnh thao tác cơ sở dữ liệu.

Dữ liệu được đưa vào vùng đệm có thể là xâu ký tự hoặc số 1, 2, 4, 8 byte.

Định dạng của vùng đệm truyền tham số như sau:

Phần tử 1	Phần tử 2	. . .	

Mỗi phần tử i có cấu trúc như sau:

Type	Len	Data

Trong đó Type có kích thước 1 byte xác định kiểu của dữ liệu : Xâu hoặc số

Len có kích thước 1 byte xác định độ dài của dữ liệu (độ dài của trường Data).

Data là dữ liệu thực sự.

4.3.1.6 Vùng đệm lệnh

Vùng đệm dùng để chứa các lệnh giả định được nạp từ file phục vụ cho việc thực thi chương trình. Lúc khởi tạo, con trỏ lệnh được trỏ về đầu vùng đệm để đọc lệnh đầu tiên.

4.3.2 Bộ lệnh của máy ảo

Mỗi lệnh bao gồm mã lệnh, các thông tin điều khiển, các toán hạng nếu có.

Kích thước của mỗi lệnh phụ thuộc vào mã lệnh và các thông tin điều khiển

Mã lệnh xác định lệnh đó thực hiện nhiệm vụ gì hay hành vi thao tác với dữ liệu.

Các thông tin điều khiển dùng để xác định kích thước dữ liệu hoặc cách xác định địa chỉ của dữ liệu.

Toán hạng có thể là giá trị thực sự hoặc địa chỉ.

Mỗi lệnh sẽ có một mã lệnh riêng, mã lệnh này tương ứng với các giá trị từ 0 đến

255. Mã lệnh này được chứa trong một byte.Kích thước của mỗi lệnh phụ thuộc vào mã lệnh, các lệnh được chia thành các nhóm sau :

- Lệnh số học như cộng, trừ , nhân, chia các số nguyên, thực
- Lệnh so sánh
- Lệnh nhảy.
- Lệnh liên quan đến ngăn xếp
- Lệnh vào ra.
- Lệnh thao tác với cơ sở dữ liệu của các ma trận.
- Lệnh gọi hàm thư viện tính toán.

4.3.2.1 Các lệnh số học

Gồm các lệnh cộng, trừ, nhân, chia thực hiện trên hai toán hạng. Các toán hạng này có thể có kích thước là 1, 2, 4, 8 byte.

Độ dài của lệnh phụ thuộc vào kích thước của toán hạng.

Định dạng của lệnh như sau

Mã phép toán	**Lựa chọn**	**Toán hạng 1**	**Toán hạng 2**	**Kết quả**

Mã phép toán có kích thước 1 byte với các giá trị và ý nghĩa như sau:

Giá trị	Ký hiệu	Ý nghĩa
$00	Add	Phép cộng
$01	Sub	Phép trừ
$02	Mul	Phép nhân
$03	DivideReal	Phép chia số thực
$04	Div	Phép chia nguyên , lấy thương
$05	Mod	Phép chia nguyên , lấy số dư

Bảng 4-1: Bảng giá trị của lệnh số học

Lựa chọn có kích thước 2 byte (bao gồm lựa chọn1 và lựa chọn 2).

4 bit cao của byte cao (lựa chọn1) cho biết chế độ địa chỉ với các giá trị như trong bảng sau:

Giá trị	Ý nghĩa
0	Toán hạng 1 và 2 là giá trị
1	Toán hạng 1 là giá trị, toán hạng 2 là địa chỉ
2	Toán hạng 1 là địa chỉ , toán hạng 2 là giá trị
3	Toán hạng 1 và 2 là địa chỉ

Bảng 4-2: Bảng giá trị xác định chế độ địa chỉ trong lệnh số học

4 Bit thấp của byte cao (lựa chọn 1) xác định kích thước của các toán hạng trong trương hợp toán hạng là giá trị. Nếu toán hạng là địa chỉ thì nó có kích thước 4 byte.

Trường lựa chọn 2 chưa được dùng.

Kết quả là một địa chỉ 4 byte.

4.3.2.2 Lệnh so sánh

Lệnh này thực hiện phép so sánh hai toán hạng với nhau, tác động lên biến cờ.
Định dạng của lệnh như sau

Mã phép so sánh	Lựa chọn1	Lựa chọn2	Toán hạng 1	Toán hạng 2

Mã phép so sánh có giá trị và ý nghĩa như trong bảng sau:

Giá trị	Ký hiệu	Ý nghĩa
$06	Gtr	So sánh >
$07	Geq	So sánh >=
$08	Lss	So sánh <
$09	Leq	So sánh <=
$0A	Equ	So sánh =
$0B	Neq	So sánh <>

Bảng 4-3: Bảng giá trị của lệnh so sánh

Lựa chọn 1 và lựa chọn 2 đều có kích thước 1 byte và có ý nghĩa như sau:
Giá trị 4 bit cao của lựa chọn 1 có ý nghĩa được mô tả trong bảng sau:

Giá trị	Ý nghĩa
0	Toán hạng 1 và 2 là giá trị
1	Toán hạng 1 là giá trị, toán hạng 2 là địa chỉ
2	Toán hạng 1 là địa chỉ , toán hạng 2 là giá trị
3	Toán hạng 1 và 2 là địa chỉ

Bảng 4-4: Bảng giá trị xác định chế độ địa chỉ trong lệnh so sánh

Giá trị của 4 bít thấp của lựa chọn 1 xác định kích thước của toán hạng trong trường hợp toán hạng là giá trị (nếu 2 toán hạng là giá trị thì chúng phải có cùng kích thước).

Trường lựa chọn 2 chưa sử dụng.

Hai Toán hạng nếu là địa chỉ thì nó có kích thước 4 byte.

Lệnh này tác động vào thanh ghi cờ, nếu kết quả so sánh là đúng thì giá trị của thanh ghi cờ bằng 1 và nếu ngược lại thì giá trị của thanh ghi cờ bằng 0.

4.3.2.3 Các lệnh nhảy

Gồm lệnh nhảy không điều kiện và nhảy có điều kiện.

Lệnh nhảy có điều kiện có hai lệnh đó là lệnh nhảy đúng và lệnh nhảy sai.

Lệnh nhảy đúng sẽ thực hiện khi kết quả của một biểu thức logic là đúng (tức là khi thanh ghi cờ có giá trị bằng 1).

Lệnh nhảy sai sẽ thực hiện khi kết quả của một biểu thức logic là sai (tức thanh ghi cờ có giá trị bằng 0).

Định dạnh của lệnh như sau

Mã lệnh	Lựa chọn 1	Lựa chọn 2	Địa chỉ mới

Mã lệnh có giá trị và ý nghĩa được mô tả trong bảng sau:

Giá trị	Ký hiệu	Ý nghĩa
$0C	JMP	Nhảy không điều kiện
$0D	JT	Nhảy có điều kiện , khi cờ bằng 1
$0E	JF	Nhảy điều kiện, khi cờ bằng 0

Bảng 4-5: Bảng giá trị của lệnh nhảy

Nếu 4 bit cao của Lựa chọn 1 bằng 0 thì Địa chỉ mới có kích thước 4 byte xác định vị trí nhảy đến của con trỏ chương trình PC.

Trong trường hợp 4 bit cao của lựa chọn 1 bằng 1 thì Địa chỉ mới có kích thước 4 byte xác định địa chỉ của biến tạm nguyên , biến này xác định vị trí nhảy đến của con trỏ chương trình PC.

4.3.2.4 Các lệnh liên quan đến ngăn xếp

Gồm hai lệnh Push và Pop.
Định dạng của lệnh như sau

Mã lệnh	Lựa chọn 1	Lựa chọn 2	Toán hạng

Mã lệnh có giá trị và ý nghĩa được xác định trong bảng sau:

Giá trị	Ký hiệu	Ý nghĩa
$0F	PUSH	Đẩy dữ liệu vào ngăn xếp
$10	POP	Lấy dữ liệu ra khỏi ngăn xếp

Bảng 4-6: Bảng giá trị của lệnh thao tác ngăn xếp

Lựa chọn 1 và lựa chọn 2 đều có kích thước 1 byte với giá trị và ý nghĩa được mô tả như sau:

- 4 bit cao của lựa chọn 1 xác định cách tính địa chỉ toán hạng với các giá trị:
  - 0 : toán hạng là giá trị trực tiếp.
  - 1 : toán hạng là địa chỉ.
  - 2 : toán hạng gồm 3 trường xác định 1 phần tử của ma trận, mỗi trường có kích thước 4 byte. Trường thứ nhất xác định địa chỉ biến ma trận trong bộ nhớ máy ảo nói cách khác đây là địa chỉ của ngăn nhớ 4 byte chứa địa chỉ thực sự (không phải địa chỉ ảo) của ma trận hay mảng 2 chiều. Trường thứ 2 và thứ 3 là địa chỉ của 2 biến trung gian 4 byte trong bộ nhớ máy ảo nơi chứa giá trị của chỉ số hàng và cột của phần tử trong ma trận cần thao tác.
  - 3 : toán hạng gồm 2 trường xác định 1 phần tử của vector, mỗi trường có kích thước 4 byte. Trường thứ nhất là địa chỉ của biến vector (con trỏ) hay nó là địa chỉ của ngăn nhớ 4 byte trong bộ nhớ máy ảo nơi chứa địa chỉ thực sự (không phải địa chỉ ảo) của vector hay mảng 1 chiều.Trường thứ hai là địa chỉ của một biến trung gian trong bộ nhớ máy ảo nơi chứa chỉ số của phần tử vector cần thao tác.
- 4 bit cao của lựa chọn 2 xác định kích thước thực sự của toán hạng.
- 4 bit thấp của lựa chọn 2 xác định kích thước cần đẩy hoặc lấy ra từ ngăn xếp.

Toán hạng là địa chỉ nơi cất dữ liệu được lấy ra từ ngăn xếp hoặc địa chỉ của dữ liệu sẽ được cất vào ngăn xếp hoặc là một giá trị trực tiếp (trong trường hợp lệnh PUSH).

4.3.2.5 Lệnh gán giá trị.

Đây là lệnh gán giá trị của một vùng nhớ cho một vùng nhớ khác.

Định dạng của lệnh như sau

Mã lệnh	Lựa chọn1	Lựa chọn2	Toán hạng 1	Toán hạng 2

Mã lệnh ký hiệu là MOVE có độ dài 1 byte và giá trị bằng $60.

Lựa chọn 1 có độ dài 1 byte xác định chế độ địa chỉ của toán hạng 1 và toán hạng 2 trong đó:

-4 bít cao của Lựa chọn 1 xác định chế độ địa chỉ của toán hạng 1 (toán hạng nguồn) với giá trị và ý nghĩa giống như lệnh PUSH.

-4 bít thấp của Lựa chọn 1 có giá trị bằng 1 xác định toán hạng 4 là một địa chỉ.

4.3.2.6 Các lệnh vào ra File

Đây là hai lệnh đọc dữ liệu kiểu ma trận hoặc ghi dữ liệu ma trận ra một file Text

File này có định dạng như sau :

- Ma trận : Dòng đầu ghi 2 số nguyên M, N xác định kích thước hàng, cột của ma trận
- Trong M dòng tiếp theo, mỗi dòng ghi N số thực, các số cách nhau ít nhất 1 dấu cách.
- Vector : Dòng đầu ghi số nguyên M xác định kích thước của vector, dòng thứ hai ghi M số thực(các thành phần của vector) cách nhau bởi dấu cách.

Định dạng của lệnh như sau

Mã lệnh	Lựa chọn 1	Lựa chọn 2

Mã lệnh có giá trị và ý nghĩa được mô tả trong bảng sau:

Giá trị	Ký hiệu	Ý nghĩa
$11	READFROMFILE	Đọc dữ liệu từ file
$12	WRITETOFILE	Ghi dữ liệu ra file

Bảng 4-7: Bảng giá trị của lệnh vào ra file.

Lựa chọn 1, lựa chọn 2 chưa được sử dụng.

Các tham số khác của lệnh vào ra qua file nằm trong vùng đệm truyền tham số.

4.3.2.7 Các lệnh thao tác dữ liệu trong cơ sở dữ liệu

Các lệnh này chỉ có mã lệnh, các tham số được lấy từ vùng đệm truyền tham số. Số lượng cũng như kiểu của tham số phụ thuộc vào mã lệnh.

Định dạng của lệnh như sau

Mã lệnh	Lựa chọn 1	Lựa chọn 2

Mã lệnh có giá trị và ý nghĩa được mô tả trong bảng sau:

Giá trị	Ký hiệu	Ý nghĩa
$27	MatGetSize	Lấy kích thước của ma trận trong cơ sở dữ liệu
$28	VecGetSize	Lấy kích thước vector trong cơ sở dữ liệu
$2F	SelectRow	Chọn ra một số hàng của ma trận trong cơ sở dữ liệu
$30	SelectCol	Chọn ra một số cột của ma trận trong cơ sở dữ liệu
$31	SelectRowCol	Chọn ra một phần của ma trận trong cơ sở dữ liệu
$32	Store	Lưu trữ ma trận, vector từ bộ nhớ vào cơ sở dữ liệu
$20	CreateMatrix	Khởi tạo ma trận trên bộ nhớ
$21	CreateVector	Khởi tạo vector trên bộ nhớ
$27	DestroyMatrix	Hủy ma trận trên bộ nhớ
$28	DestroyVector	Hủy vector trên bộ nhớ

Bảng 4-8: Bảng giá trị của lệnh thao tác dữ liệu

4.3.2.8 Các lệnh nạp tham số vào vùng đệm

Tham số có hai loại là xâu ký tự và số. Các lệnh đưa tham số ra vùng đệm đối với hai loại này có định dạng khác nhau.

- Lệnh thêm tham số là xâu ký tự có định dạng như sau :

Mã lệnh	Lựa chọn 1	Lựa chọn 2	xâu ký tự

trong đó mã lệnh ký hiệu là ADDSTRINGPARA, có giá trị là $23.Trường Lựa chọn 2 chứa độ dài của xâu ký tự.

-Lệnh thêm tham số là giá trị số có định dạng như sau:

Mã lệnh	Lựa chọn 1	Lựa chọn 2	Toán hạng

Trong đó mã lệnh ký hiệu là ADDNUMBERPARA và có giá trị là $22. Trường lựa chọn và toán hạng có giá trị và ý nghĩa như trong lệnh PUSH và POP.

4.3.2.9 Lệnh khởi tạo lại vùng đệm tham số

Mã lệnh ký hiệu là RefreshParaBuffer và có giá trị là $24, lệnh này không có toán hạng đi cùng.Trường Lựa chọn 2 chứa số lượng tham số sẽ đưa vào vùng đệm tham số.

4.3.2.10 Lệnh gọi thư viện tính toán

Định dạng của lệnh giống với định dạng của lệnh thao tác cơ sở dữ liệu.

Mã lệnh ký hiệu là Exec và có giá trị là $50.

4.3.2.11 Lệnh hiển thị dữ liệu ra màn hình

Lệnh này thực hiện hiển thị các dữ liệu ra màn hình. Dữ liệu này có thể là xâu ký tự, số nguyên, thực. Các dữ liệu này nằm trong vùng đệm truyền tham số.

Định dạng của lệnh như sau

Mã lệnh	Lựa chọn 1	Lựa chọn 2

Mã lệnh có giá trị và ý nghĩa được mô tả trong bảng sau:

Giá trị	Ký hiệu	Ý nghĩa
$25	WRITE	Ghi bình thường, không xuống dòng
$26	WRITELN	Ghi xong xuống dòng

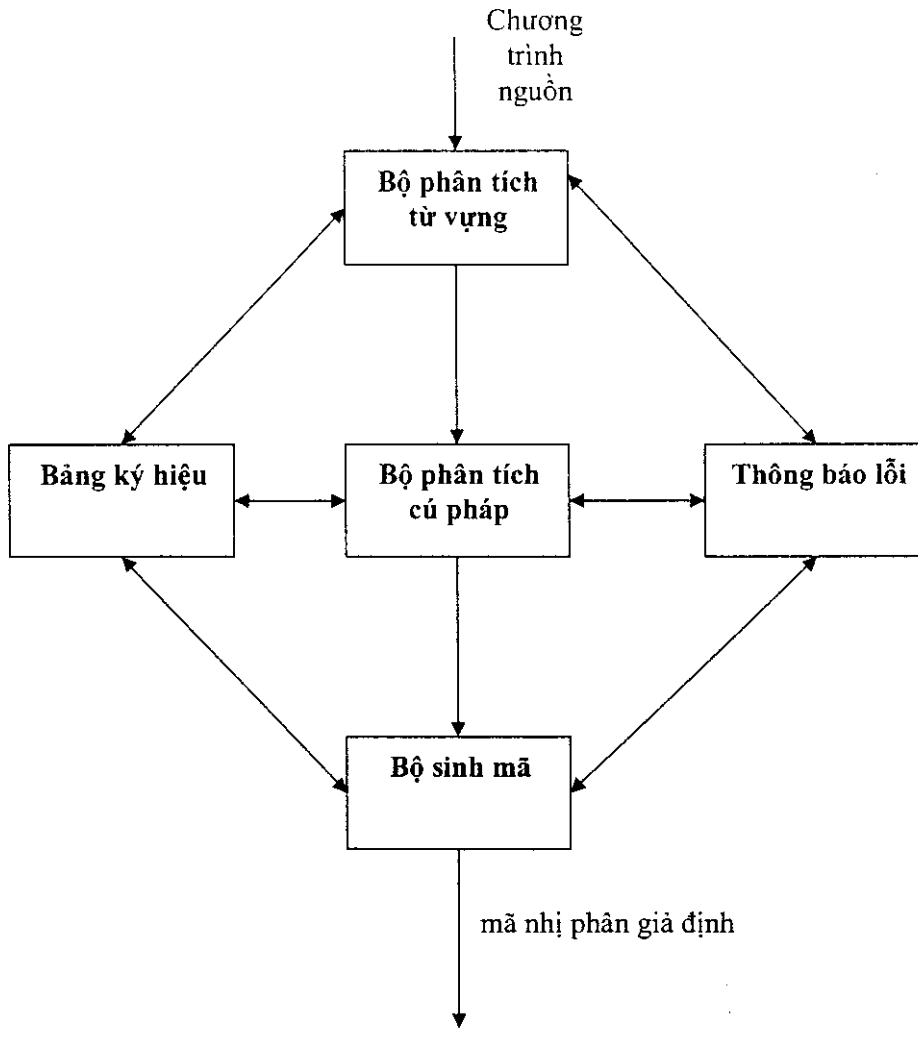
Bảng 4-9: Bảng giá trị của lệnh hiển thị dữ liệu ra màn hình.

4.4 Xây dựng bộ dịch

Quá trình biên dịch gồm 3 phase chính:

- Phân tích từ vựng.
- Phân tích cú pháp.
- Sinh mã.

về chi tiết có thể minh hoạ theo sơ đồ sau:



Hình 4-4 Sơ đồ 3 pha của bộ biên dịch.

Việc phân chia theo từng phase để tìm được công cụ mạnh và giải pháp thích hợp đối với từng phase, còn khi cài đặt có thể kết hợp một số phase lại thành 1 số module chương trình.

4.4.1 Bộ phân tích từ vựng

Pha phân tích từ vựng được thực hiện bởi Otomat hữu hạn.

Chương trình nguồn, qua bộ phân tích từ vựng sẽ cho ra các từ vựng thuộc 1 trong các loại sau:

- Từ khóa, tên chuẩn
- Tên
- Hằng số, hằng xâu

- Dấu (dấu đơn, dấu kép)

4.4.2 Bộ phân tích cú pháp

Cho văn phạm G = (N, T, S, P) sinh ra ngôn ngữ $L(G) = \{\omega \in T^* \mid S \Rightarrow^* \omega\}$

Phân tích câu ω là xét xem ω có thuộc L(G) hay không.

Có hai khả năng xảy ra:

$\omega \in L(G) \rightarrow$ theo nghĩa hẹp là chương trình viết đúng cú pháp.

$\omega \notin L(G) \rightarrow$ theo nghĩa hẹp là chương trình viết sai cú pháp.

Như vậy phân tích câu ω là biết cách xây dựng cây cú pháp của ω.

4.4.2.1 Các phương pháp phân tích

Có hai phương pháp phân tích đó là phân tích trên xuống (Top-down) và phân tích dưới lên (Bottom-up).

Có hai kiểu phân tích đó là phân tích quay lui và phân tích tất định.

Trong nội dung của luận văn này chỉ xét phương pháp phân tích Top-down tất định cụ thể là phương pháp phân tích đệ quy trên xuống.

Trong phương pháp này, xâu vào được xem xét một lượt từ trái qua phải, tại mỗi bước người ta xác định một sản xuất duy nhất có thể có.

Văn phạm LL(1)
Cho văn phạm G = (N, T, S, P), số nguyên dương k, $\alpha \in V^*$.

Ta đưa vào hai tập hợp sau:

$$FIRSTk(\alpha) = \{x \mid x \in T,\ \alpha =>^* x\beta \text{ và } l(x) = k \text{ hoặc } \alpha =>^* x \text{ và } l(x) \leq k\}$$

FIRSTk(α) là tập các xâu x chứa k ký hiệu kết thúc đầu tiên suy dẫn từ α. Nếu α chỉ gồm các ký hiệu kết thúc, ta lấy k ký hiệu đầu.

$$FOLLOWk(\alpha) = \{x \mid x \in T^*,\ S =>^* \beta\alpha\delta \text{ và } x \in FIRSTk(\delta)\}.$$

FOLLOWk(α) là tập các xâu x, chứa k ký hiệu kết thúc, đứng ngay bên phải α trong bất kỳ dạng câu nào đó, đặc biệt $\alpha = A \in N$, và βA là dạng câu thì

$FOLLOW1(A) = \{\varepsilon\}$.

Điều kiện để văn phạm là LL(1)

Văn phạm G = {N, T, S , P} là LL(1) khi và chỉ khi mỗi quy tắc P có dạng A -> $\alpha 1 \mid \alpha 2 \mid \ldots \mid \alpha n$ ($n \geq 2$) nó thoả mãn các điều kiện sau:

a.FIRST(αi) $\cap$FIRST(αj) = $\varnothing$, $1 \leq i < j \leq n$.

b. Nếu có $\alpha i \Rightarrow^* \epsilon$ thì phải có FIRST1(αj) $\cap$ FOLLOW1(A) = $\varnothing$ (với mọi $j \neq i$, $1 \leq j \leq n$).

Đối chiếu với văn phạm của ngôn ngữ lập trình đề ra, ta thấy văn phạm đó chưa thỏa mãn điều kiện LL(1) ví dụ quy tắc:

<câulệnh> → if <dk> then<câulệnh>| if <dk>then <câulệnh>else<câulệnh>

Văn phạm này cần được chuyển về văn phạm LL(1) tương đương.

Việc chuyển về văn phạm tương đương này được thực hiện bằng phương pháp nhóm thừa số chung.

Ví dụ 4.1. Văn phạm S -> $a\alpha \mid a\beta$

vế phải của hai sản xuất đều được bắt đầu bằng ký hiệu kết thúc a, như vậy từ ký hiệu không kết thúc S và 1 ký hiệu xem xét là a thì không thể quyết định được sẽ áp dụng sản xuất nào trong hai sản xuất trên.

Văn phạm trên có thể chuyển về văn phạm tương đương bằng cách đưa ký hiệu kết thúc a làm thừa số chung, văn phạm tương đương như sau:

S -> aA

A -> $\alpha \mid \beta$

Tiếp tục áp dụng nhóm thừa số chung đối với hai xâu α và β nếu chúng vẫn còn có các tiền tố (ký hiệu kết thúc ở bên trái nhất) giống nhau.

Trong ngôn ngữ lập trình quy tắc của câu lệnh rẽ nhánh được chuyển về dạng LL(1) tương đương như sau:

<câulệnh> -> if <dk> then <câulệnh> <mệnh đề tiếp>

<mệnh đề tiếp> -> ϵ | else <câulệnh>

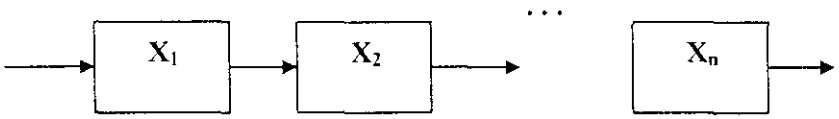
4.4.2.2 Áp dụng phương pháp phân tích đệ quy trên xuống

Cho văn phạm G = (N, T, S, P) là LL(1) và xâu ω. Tư tưởng chủ đạo của giải thuật phân tích đệ quy trên dưới là với mỗi ký hiệu A $\in$ N, ta xây dựng một thủ tục tương ứng. Quá trình phân tích bắt đầu là thủ tục ứng với ký hiệu S (ký hiệu khởi đầu) và gọi tới các thủ tục khác một cách đệ quy.

Để thuận tiện ta chuyển quy tắc A → $\alpha 1 \mid \alpha 2 \mid \ldots \mid \alpha k$ dạng BNF thành sơ đồ cú pháp theo các qui định đã được nêu ở phần xây dựng ngôn ngữ lập trình.

Khi có sơ đồ cú pháp tương ứng với quy tắc của văn phạm đã cho, ta đưa ra một số qui định để diễn dịch sơ đồ thành chương trình tương ứng. Để làm điều đó, ta ký hiệu câu lệnh diễn dịch sơ đồ S là T(S). Các qui định sau:

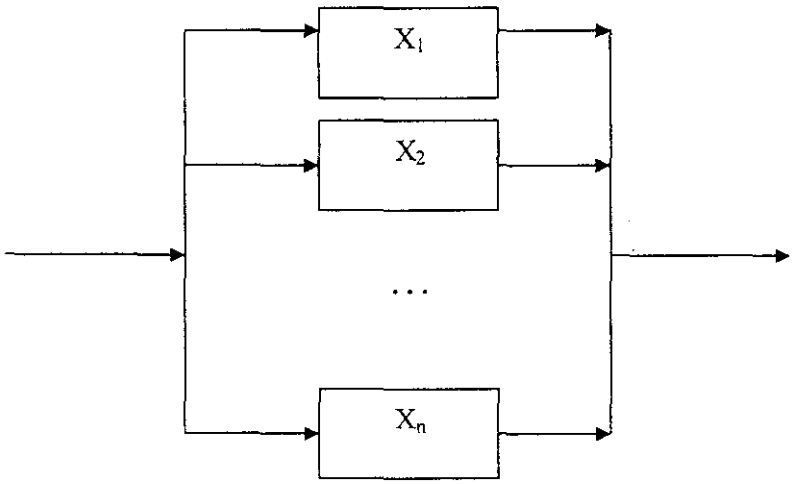
1- Nếu sơ đồ cú pháp có dạng:



thì diễn dịch tương ứng với câu lệnh ghép (tựa pascal):

```
begin
      T(X1) ;
      T(X2) ;
      . . . .
      T(Xn)
end
```

2- Nếu sơ đồ cú pháp có dạng:

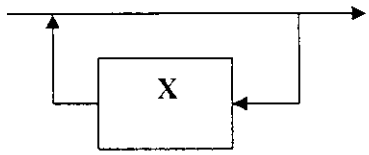


thì diễn dịch tương ứng câu lệnh chọn (tựa pascal):

```
if ch in D1 then T(X1) else
if ch in D2 then T(X2) else
. . .
if ch in Dn then T(Xn) else SAI
```

trong đó Dj = FIRST1(Xj) (1 ≤ j ≤ n)

3- Nếu sơ đồ cú pháp có dạng:

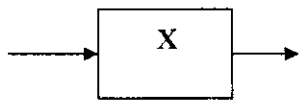


thì diễn dịch thành câu lệnh:

while ch in D do T(X)

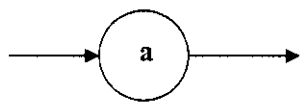
trong đó D = FIRST1(X), và với T(X) ta áp dụng 1 đến 5.

4- Nếu sơ đồ cú pháp có dạng:



thì ứng với câu lệnh gọi thủ tục T(X) ($X \in N$).

5- Nếu sơ đồ cú pháp: có dạng



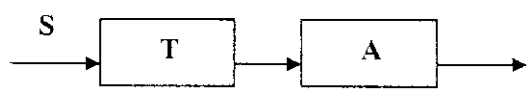
thì ứng với câu lệnh:

if ch = a then DOC else SAI

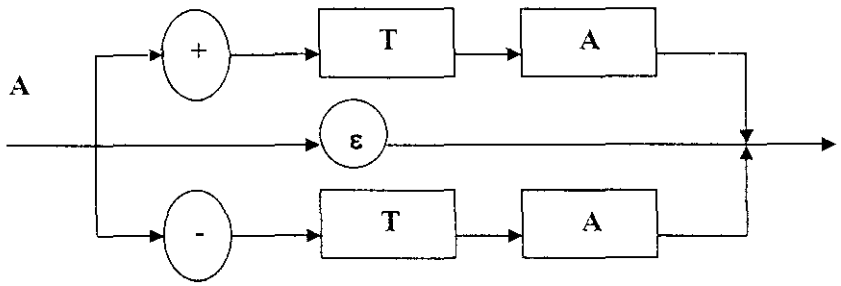
trong đó DOC và SAI là hai thủ tục làm nhiệm vụ đọc ký hiệu tiếp theo trên xâu vào và thông báo lỗi đã được xây dựng sẵn.

Ví dụ 4.2. Xét văn phạm G = (N, T, P, S) trong đó
N = {S, T, A, B, F}, T = { +, - , * , / , (,) , a , ϵ}
P bao gồm các sản xuất:
S -> TA
A -> +TA | -TA | ϵ
T -> FB
B -> *FB | /FB | ϵ
F -> (S) | a.
Sơ đồ cú pháp của văn phạm trên như sau:

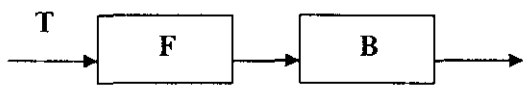
Sơ đồ S.



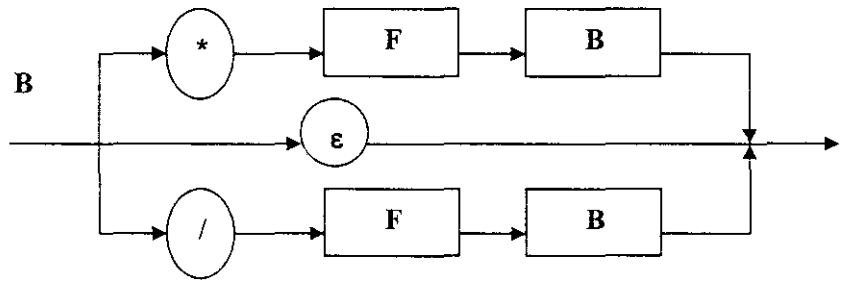
Sơ đồ A.



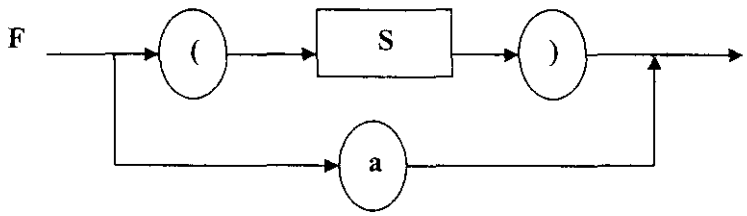
Sơ đồ T.



Sơ đồ B.



Sơ đồ F.



Ứng với mỗi ký hiệu kết thúc, xây dựng các thủ tục đoán nhận như sau:

```
procedure S;
begin
      T;
      A;
```

```
end;

procedure A;

begin

      if ch = '+' then

            begin

                  GetCh;

                  T;

                  A;

            end

      else if ch = '-' then

            begin

                  GetCh;

                  T;

                  A;

            end;

end;


procedure T;

begin

      F;

      B;

end;


procedure B;

begin

      if ch = '*' then

            begin

                  GetCh;
```

```
                F;
                B;
          end
     else if ch = '/' then
          begin
                GetCh;
                F;
                B;
          end;
end;

procedure F;
begin
     if ch = '(' then
          begin
                GetCh;
                S;
                if ch = ')' then
                     GetCh
                else Error('Thieu dau ngoac');
          end
     else if ch = 'a' then
          GetCh
     else Error('Loi trong cau lenh');
end;
```

Chương trình chính phân tích cú pháp đệ quy trên xuống:

```
begin
```

```
        LoadProgram(str);
        GetCh;
        S;
End
```

Trong các thủ tục trên, LoadProgram, GetCh và Error là các thủ tục đã được xây dựng sẵn, thủ tục LoadProgram có nhiệm vụ nạp chương trình nguồn vào xâu str, thủ tục GetCh có nhiệm vụ đọc ký hiệu vào tiếp theo và thủ tục Error có nhiệm vụ lưu trữ tất cả các thông tin về lỗi.

4.4.2.3 Chuẩn hoá cây suy dẫn

Cây suy dẫn cần được chuẩn hoá để tạo thuận tiện cho việc sinh mã.

Chuẩn hoá cây biểu thức.

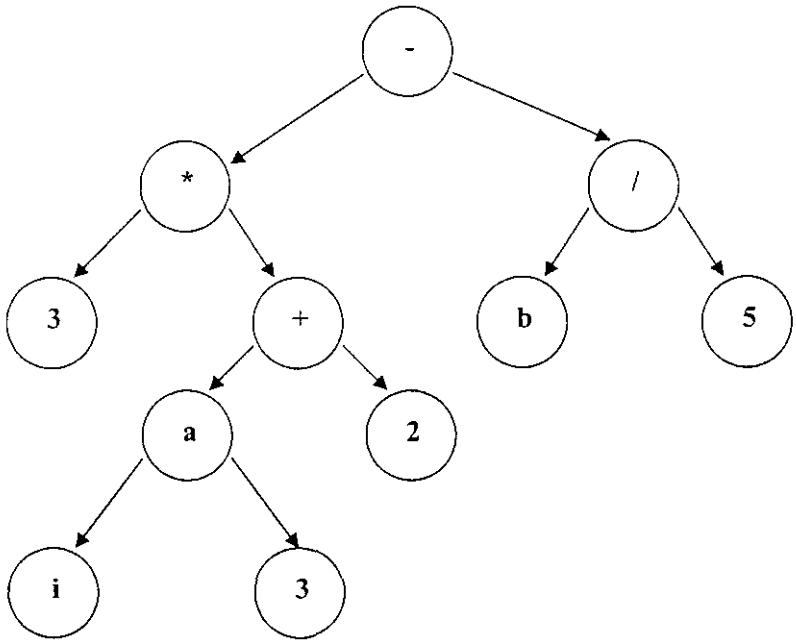
Biểu thức đơn giản nhất chỉ là một biến (không phải là phần tử của mảng 1 chiều hoặc 2 chiều) hoặc một hằng số, trong trường hợp này không cần chuẩn hoá.

Trong trường hợp biểu thức là một phần tử của mảng 1 chiều hoặc 2 chiều thì cây chuẩn hóa được tạo thành bởi nút gốc là tên biến mảng, các nút con(1 hoặc 2 nút con) là các cây biểu thức đã được chuẩn hóa ứng với các chỉ số.

Nếu biểu thức là một lời gọi chương trình con thì cây biểu thức chuẩn hoá được tạo thành bởi nút gốc ứng với tên chương trình con, các nút con từ trái qua phải lần lượt là các biểu thức đối số đã được chuẩn hoá.

Nếu biểu thức E1 op E2 trong đó op là một toán tử số học, E1 và E2 là hai biểu thức số học thì cây biểu thức chuẩn hoá được tạo thành bởi nút gốc là nút toán tử, nút con trái ứng với biểu thức E1 đã được chuẩn hoá và nút con phải ứng với biểu thức E2 đã được chuẩn hoá.

Ví dụ 4.3. Với biểu thức `3*(a[i,3]+2) - b*5`, cây biểu thức chuẩn hoá như hình sau:



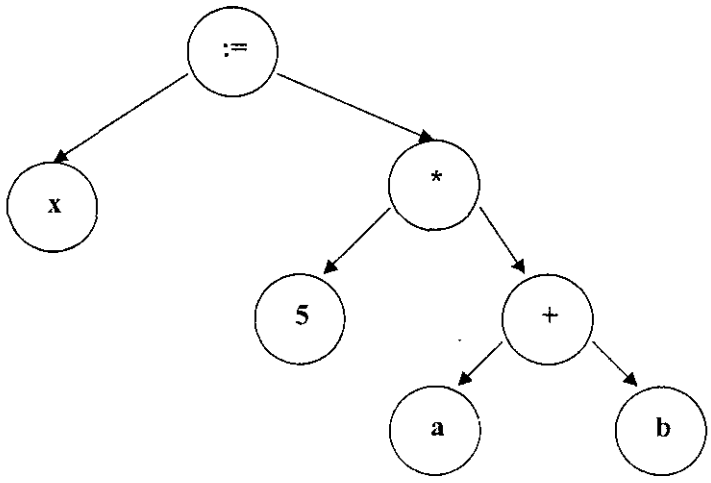
Hình 4-5 Cây biểu thức chuẩn hóa trong ví dụ 4.3

Chuẩn hoá cây suy dẫn của với câu lệnh gán

Cây suy dẫn chuẩn hoá của câu lệnh gán được toạ thành bởi:

- Nút gốc là nút ứng với toán tử gán.
- Cây con trái ứng với biến ở vế trái của câu lệnh gán.
- Cây con phải là cây đã được chuẩn hóa của biểu thức ở vế phải của câu lệnh gán.

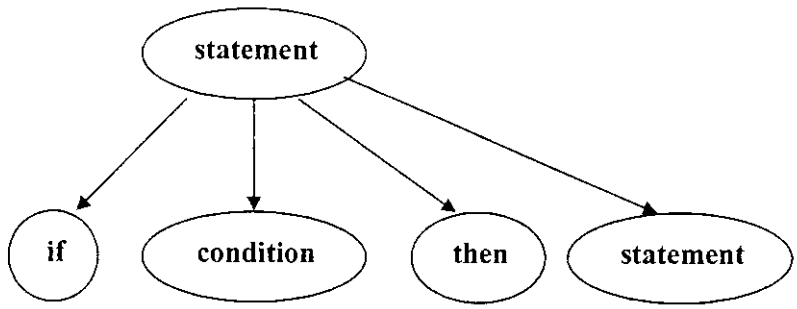
Ví dụ 4.4. Với câu lệnh gán `x := 5*(a+b)`, cây chuẩn hóa như sau



Hình 4-6 Cây cú pháp chuẩn hóa của câu lệnh gán trong ví dụ 4.4

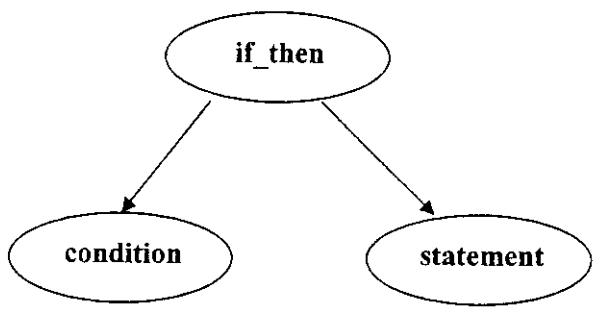
Chuẩn hoá cây suy dẫn của câu lệnh rẽ nhánh.

Cây suy dẫn của câu lệnh rẽ nhánh 1 vế có dạng sau:



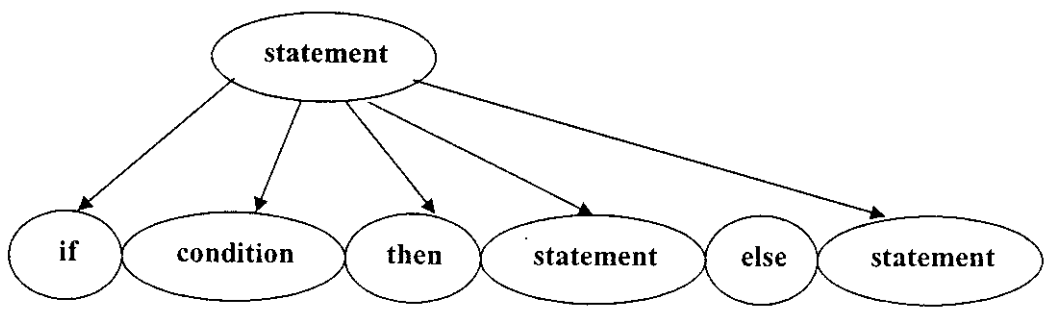
Hình 4-7 Cây suy dẫn câu lệnh rẽ nhánh 1 vế

được thu gọn về dạng sau:



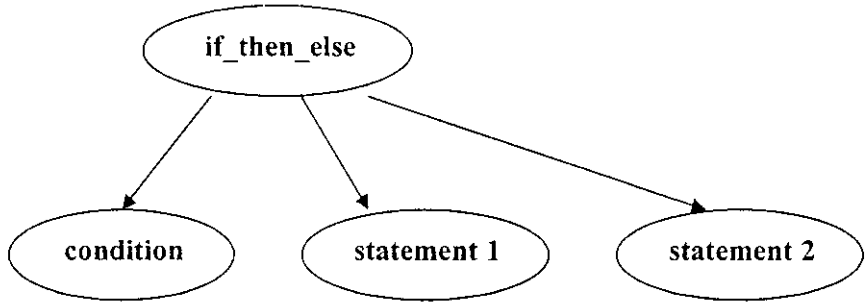
Hình 4-8 Cây suy dẫn chuẩn hóa của câu lệnh rẽ nhánh 1 vế

Cây suy dẫn câu lệnh rẽ nhánh 2 vế có dạng sau:



Hình 4-9 Cây suy dẫn câu lệnh rẽ nhánh 2 vế.

được chuẩn hoá về dạng sau:



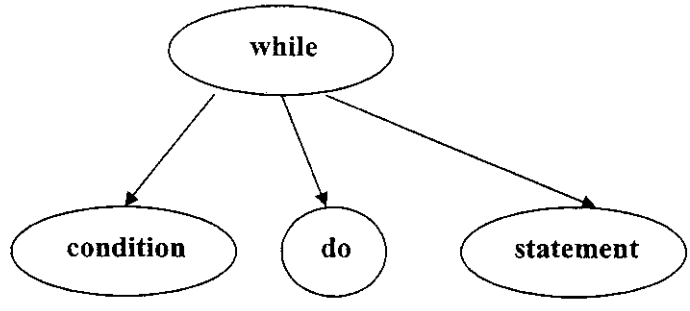
Hình 4-10 Cây suy dẫn chuẩn hóa của câu lệnh rẽ nhánh 2 về.

trong đó :

- Nút gốc có nhãn là if_then hoặc if_then_else.
- Nút điều kiện là cây biểu thức logic đã được chuẩn hoá.
- Các nút câu lệnh statement cũng đã được chuẩn hóa.

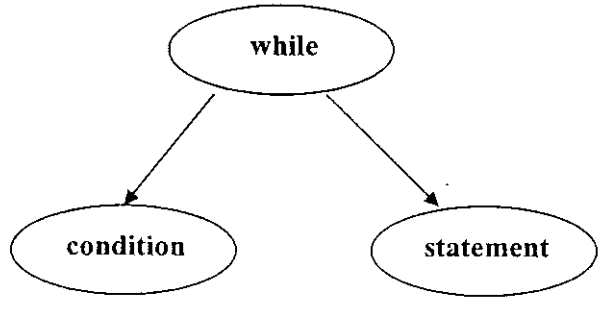
Chuẩn hoá cây suy dẫn của câu lệnh lặp.

Cây suy dẫn của câu lệnh lặp while có dạng sau:



Hình 4-11 Cây suy dẫn câu lệnh lặp while.

được chuẩn hoá về dạng sau:



Hình 4-12 Cây suy dẫn chuẩn hóa của câu lệnh lặp.

Câu lệnh lặp for được biến đổi về câu lệnh lặp while tương đương.

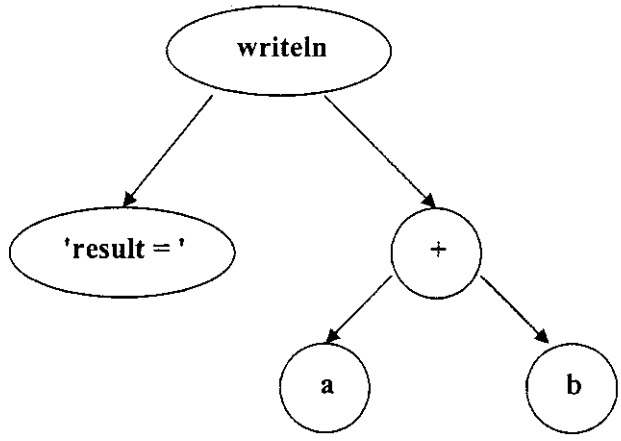
Chuẩn hoá cây suy dẫn ứng với các câu lệnh còn lại

Các câu lệnh còn lại là các câu lệnh hiển thị dữ liệu ra màn hình, lệnh gọi thư viện tính toán, câu lệnh vào ra file. . . đều có dạng như một lời gọi thủ tục. Cây suy dẫn của các câu lệnh này được chuẩn hóa về dạng sau:

Nút gốc ứng với tên của lệnh: `write, writeln, readfromfile, exec, . . .`

Các nút con lần lượt từ trái qua phải là các cây đã được chuẩn hóa của các tham số.

Ví dụ 4.5. Với câu lệnh `writeln('result = ', a+b)` được chuẩn hóa về dạng



Hình 4-13 Cây suy dẫn chuẩn hóa của câu lệnh hiển thị dữ liệu trong ví dụ 4.5

Cây cú pháp sau khi chuẩn hóa được dùng làm đầu vào cho pha sinh mã.

4.4.3 Bộ sinh mã

4.4.3.1 Bảng ký hiệu

Bảng ký hiệu được tham chiếu thường xuyên trong xử lý sinh mã cho hầu hết các nút lệnh.

Bảng ký hiệu là một cấu trúc dữ liệu dùng để lưu trữ các thông tin về các đối tượng mà chương trình dịch làm việc. Nó gồm nhiều phần tử, mỗi phần tử chứa thông tin cần thiết về tên trong chương trình nguồn. Mỗi phần tử gồm hai trường quan trọng:

- Tên: Trường này lưu trữ tên của các biến, chương trình con được khai báo và sử dụng trong chương trình nguồn.

- Thuộc tính: Trường này chứa các thông tin liên quan đến tên như kiểu, vị trí,. . .

Các thao tác đối với bảng ký hiệu

- Thêm một tên mới vào trong bảng ký hiệu.
- Kiểm tra một tên đã tồn tại trong bảng ký hiệu hay chưa.
- Tìm kiếm và lấy thông tin ứng với một tên cho trước.
- Xoá tên ra khỏi bảng ký hiệu.

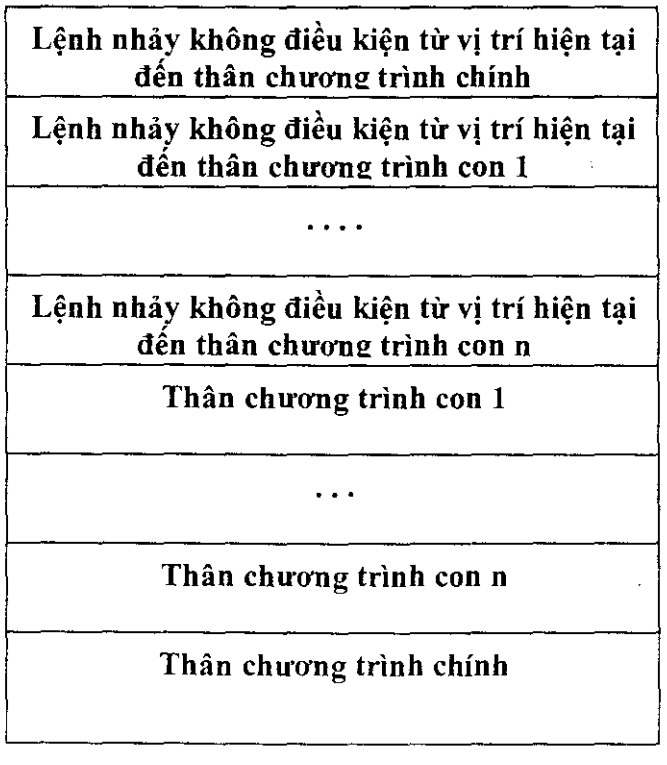
Bảng ký hiệu có thể được tổ chức theo danh sách liên kết, cây nhị phân tìm kiếm hoặc bảng băm.

Trong bộ dịch này, bảng ký hiệu được tổ chức theo danh sách LIFO (ngăn xếp). Tên mới được thêm vào đỉnh ngăn xếp và việc tìm kiếm cũng được thực hiện từ đỉnh ngăn xếp.

Trong hầu hết các câu lệnh, bảng ký hiệu đều được tham chiếu tới để tìm kiếm các thông tin liên quan đến biến như kích thước, địa chỉ. Các thông tin này được đưa vào các câu lệnh nhị phân giả định.

4.4.3.2 Cấu trúc của chương trình mã nhị phân giả định.

Cấu trúc chương trình mã nhị phân giả định được mô tả trong Hình sau:



Hình 4-14 Cấu trúc file nhị phân giả định.

4.4.3.3 Các giải thuật sinh mã cho các nút lệnh

Tất cả các câu lệnh được biến đổi đưa về dạng mã 3 địa chỉ tương ứng.

- Câu lệnh gán mã 3 địa chỉ : `A ← B op C`
- Các câu lệnh mã 3 địa chỉ khác :

```
goto L
if  A goto L
if A op B goto L.
```

Câu lệnh mã 3 địa chỉ được ánh xạ sang dãy các câu lệnh nhị phân giả định tương đương.

Ví dụ 4.6. Đoạn chương trình sau:

```
while (a > 10) and ( a<2*b+5) do a := a+b
```

được chuyển về các câu lệnh dạng 3 địa chỉ tương ứng như sau:

```
dk1:  if  a > 10 then goto dk2;
      goto Thoat;
dk2 :  t1 := 2*b;
      t2 := t1 + 5;
      if a < t2 then goto Caulenh else goto Thoat;
Caulenh:
      a := a + b;
      goto dk1;
Thoat :
```

Ví dụ 4.7. Đoạn chương trình sau

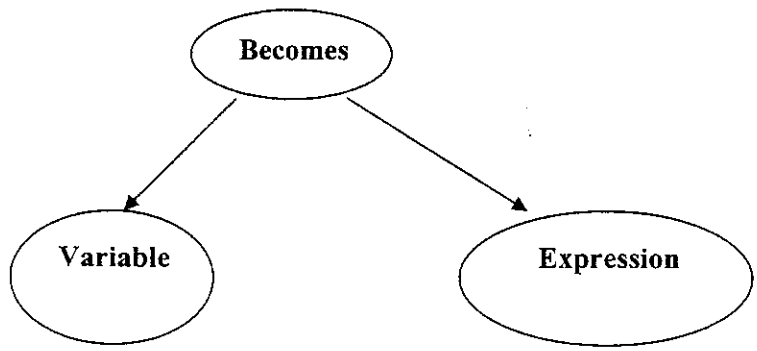
```
if  (a < 100) or ( b+1 < n) then
      a := a + 1
else begin
      a := b + n;
      b := b - 1;
end;
```

được chuyển về câu lệnh dạng 3 địa chỉ tương ứng như sau:

```
dk1 : if  a < 100 then
          goto Caulenh1;
dk2 : t1 := b+1;
     if  t1 < n then
          goto Caulenh1;
     goto Caulenh2;

Caulenh1: a := a + 1;
     goto Thoat;
Caulenh2:
     a := b +n;
     b := b - 1;
Thoat:
```

Xử lý sinh mã cho nút lệnh gán

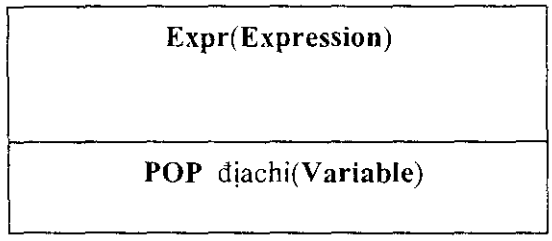
Cây cú pháp đã chuẩn hoá câu lệnh gán



Hình 4-15 Cây cú pháp của câu lệnh gán

Gọi Expr(node) là đoạn mã tính giá trị của nút biểu thức node và cất vào ngăn xếp.

Sơ đồ khối đoạn mã nhị phân giả định của câu lệnh gán như sau:



Hình 4-16 Sơ đồ khối đoạn mã của câu lệnh gán.

Xảy ra các trường hợp sau:

- Nếu node là nút lá với nhãn Id.
  - Nếu Id là một hằng số thì Expr(node) là câu lệnh: PUSH giátrị(Id).
  - Nếu Id là một biến đã có trong bảng ký hiệu thì Expr(node) là câu lệnh: PUSH địachỉ(Id).
- Nếu node là nút biểu thức dạng E1 op E2 trong đó op là toán tử số học thuộc 1 trong 6 loại (+ , - , * , / , div , mod) giả sử op là toán tử cộng (+), E1 và E2 là hai biểu thức tương ứng với 2 nút con của node, t1, t2, t3 là các biến trung gian, đoạn mã Expr(node) như sau:

  Expr(E1)

  Expr(E2)

  POP địachỉ(t2)

  POP địachỉ(t1)

  ADD địachỉ(t1), địachỉ(t2), địachỉ(t3)

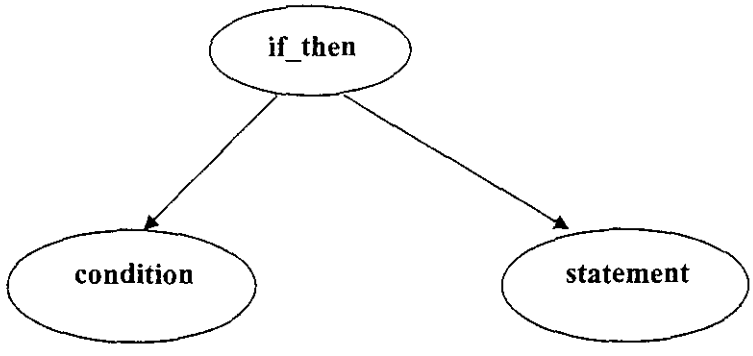
  PUSH địachỉ(t3)

- Nếu node là một lời gọi chương trình con với nhãn là Id tên của chương trình con. Các bước xử lý như sau:
  - Sinh ra các câu lệnh cất giá trị các biến này vào ngăn xếp.
  - Sinh ra các câu lệnh tính toán giá trị của các biểu thức và truyền vào cho các đối của chương trình con.
  - Cất địa chỉ của lệnh tiếp theo trong chương trình gọi vào ngăn xếp.
  - Sinh ra câu lệnh nhảy đến đầu chương trình con (từ đây chương trình con bắt đầu được thực thi).

- Sinh ra các câu lệnh khôi phục lại giá trị cho các biến trong chương trình con từ ngăn xếp.
- Sinh ra câu lệnh cất giá trị của hàm vào ngăn xếp.

Xử lý sinh mã cho nút lệnh rẽ nhánh

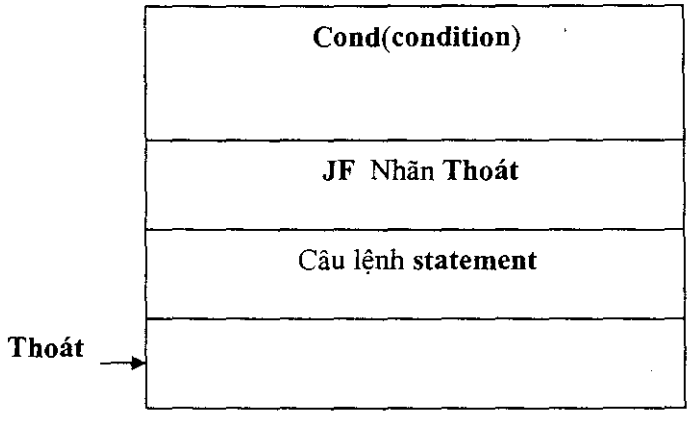
Cây cú pháp đã chuẩn hoá của câu lệnh rẽ nhánh dạng 1 vế như sau:



Hình 4-17 Cây cú pháp đã chuẩn hoá câu lệnh rẽ nhánh 1 vế

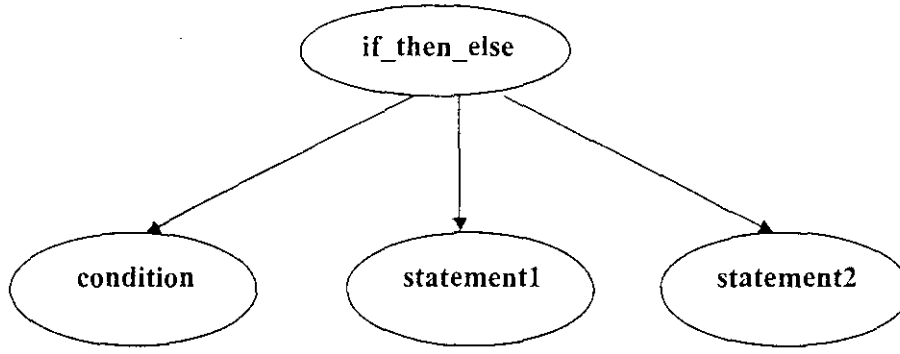
Gọi Cond(node) là đoạn mã tính giá trị của biểu thức logic node.

Sơ đồ khối đoạn mã nhị phân giả định của câu lệnh rẽ nhánh 1 vế như sau:



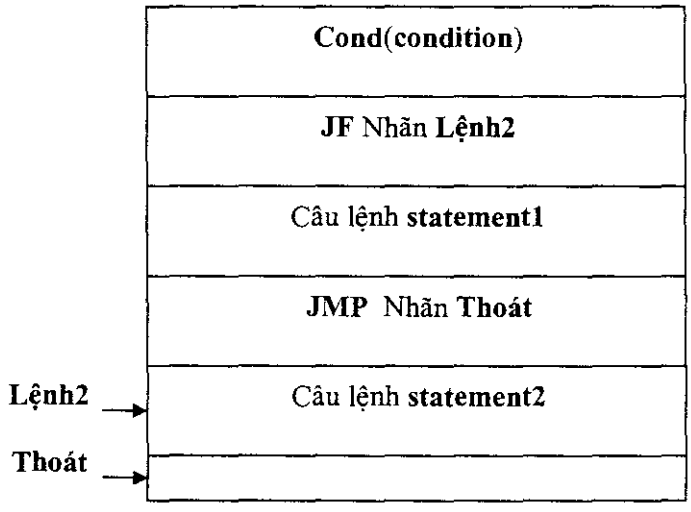
Hình 4-18 Sơ đồ khối của câu lệnh rẽ nhánh 1 vế

Cây cú pháp đã chuẩn hoá của câu lệnh rẽ nhánh 2 về như sau:



Hình 4-19 Cây cú pháp chuẩn hoá câu lệnh rẽ nhánh 2 về

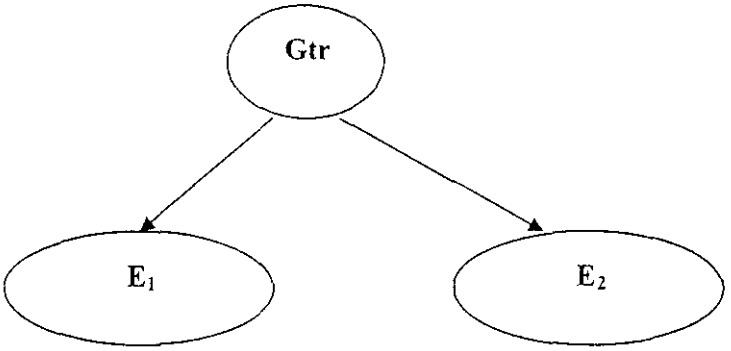
Sơ đồ khối đoạn mã nhị phân giả định của câu lệnh rẽ nhánh 2 về:



Hình 4-20 Sơ đồ khối của câu lệnh rẽ nhánh 2 về

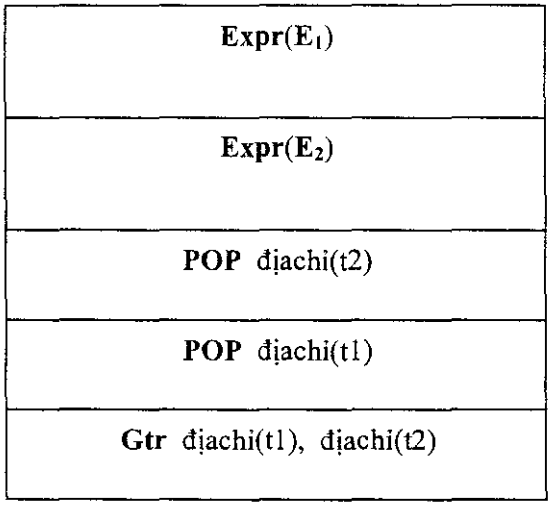
Đối với đoạn mã tính giá trị của biểu thức logic: Cond(node), xảy ra các trường hợp sau:

- Nếu node là một mệnh đề quan hệ E1 op E2 trong đó E1 và E2 là các biểu thức số học, op là toán tử quan hệ thuộc 1 trong 6 loại (= , <> , > , >= , < , <=) giả sử op là toán tử lớn hơn (>). Cây cú pháp của mệnh đề so sánh lớn hơn như sau:



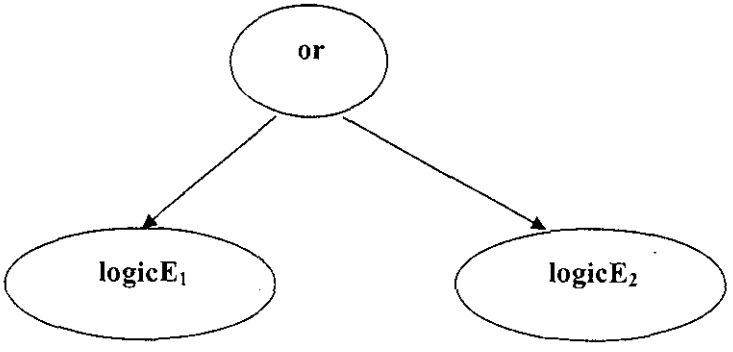
Hình 4-21 Cây cú pháp chuẩn hoá của mệnh đề quan hệ lớn hơn

Sơ đồ khối đoạn mã nhị phân giả định như sau:



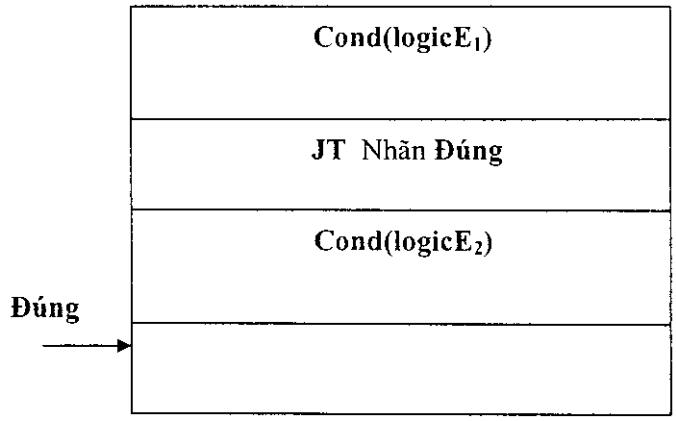
Hình 4-22 Sơ đồ khối của câu lệnh so sánh.

Nếu node là một biểu thức logic dạng logicgE1 or logicE2.



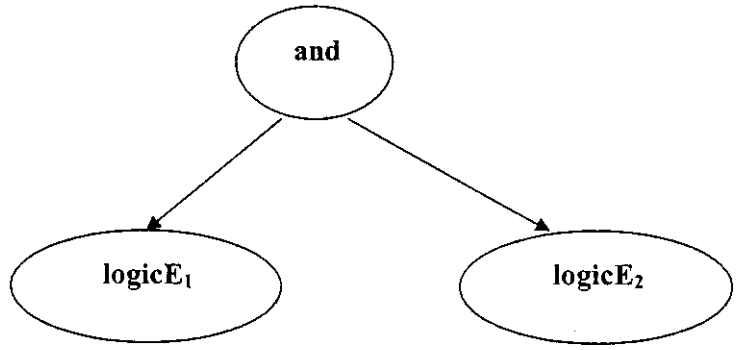
Hình 4-23 Cây cú pháp chuẩn hoá của biểu thức logic dạng OR.

Sơ đồ khối đoạn mã nhị phân giả định Cond(node) như sau:



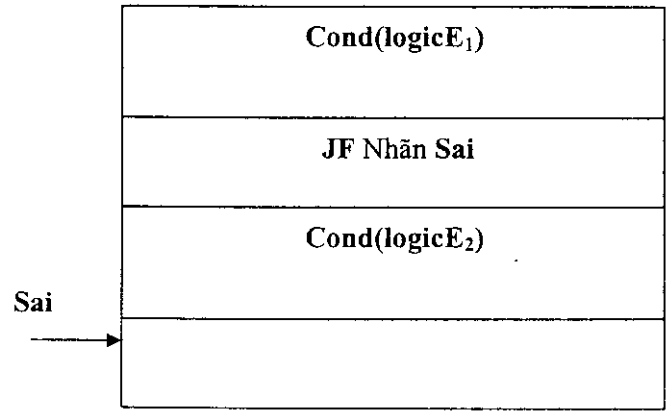
Hình 4-24 Sơ đồ khối của câu lệnh xác định giá trị của biểu thức logic OR.

Nếu node là một biểu thức logic dạng logicgE1 and logicE2.



Hình 4-25 Cây cú pháp chuẩn hoá của biểu thức logic dạng AND.

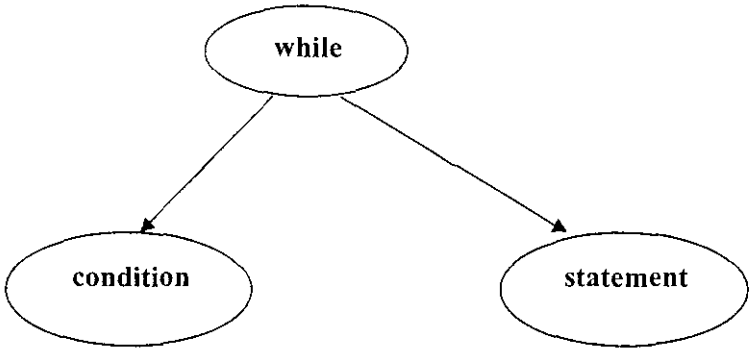
Sơ đồ khối đoạn mã nhị phân giả định Cond(node) như sau



Hình 4-26 Sơ đồ khối của câu lệnh xác định giá trị của biểu thức logic AND

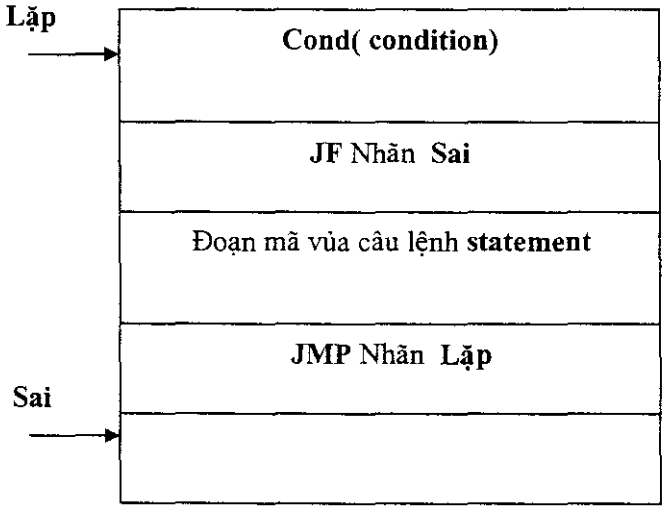
Xử lý sinh mã cho nút lệnh lặp

Cây cú pháp chuẩn hoá của câu lệnh lặp while



Hình 4-27 Cây cú pháp chuẩn hoá của câu lệnh lặp while.

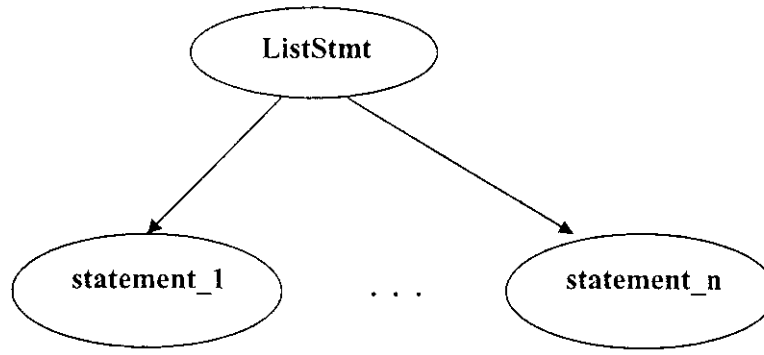
Câu lệnh lặp dạng for được đưa về dạng câu lệnh lặp while tương đương. Sơ đồ khối đoạn mã nhị phân giả định của câu lệnh lặp như sau:



Hình 4-28 Sơ đồ khối của câu lệnh lặp.

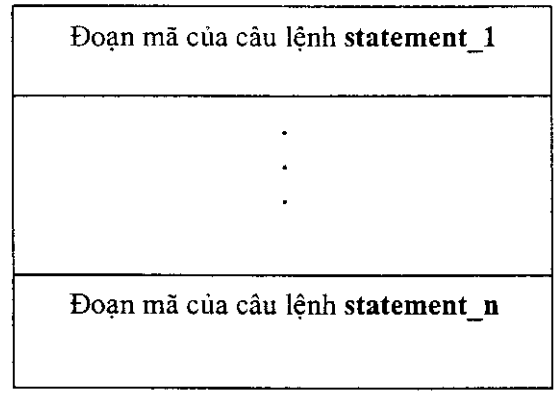
Xử lý sinh mã cho nút lệnh ghép

Cây cú pháp chuẩn hoá câu lệnh ghép như sau:



Hình 4-29 Cây cú pháp chuẩn hóa của câu lệnh ghép

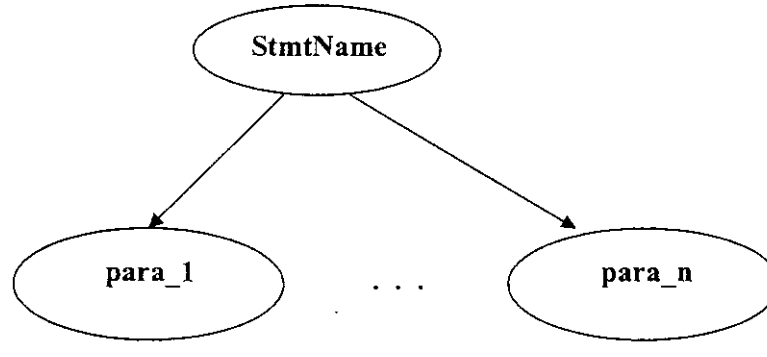
Sơ đồ khối đoạn mã nhị phân giả định của câu lệnh ghép như sau



Hình 4-30 Sơ đồ khối của câu lệnh ghép

Xử lý sinh mã cho các nút lệnh còn lại

Các câu lệnh còn lại đề có dạng lời gọi thủ tục. Cây cú pháp chuẩn hoá có dạng sau:



Hình 4-31 Cây cú pháp chuẩn hoá của các câu lệnh có dạng lời gọi thủ tục

Trong đó StmtName là tên của câu lệnh (readfromfile, exec, writeln , . . .). Các nút con para_i là các tham số thuộc 1 trong 2 loại là hằng xâu ký tự hoặc biểu thức số học.

Sơ đồ sinh mã chung được mô tả như sau:

- Lần lượt sinh các đoạn mã tính giá trị và đưa các tham số para_i ra vùng đệm truyền tham số.
- Sinh ra câu lệnh có tên StmtName tương ứng.

Ví dụ 4.8. Câu lệnh exec('matmultadd', mt, x, y, w)

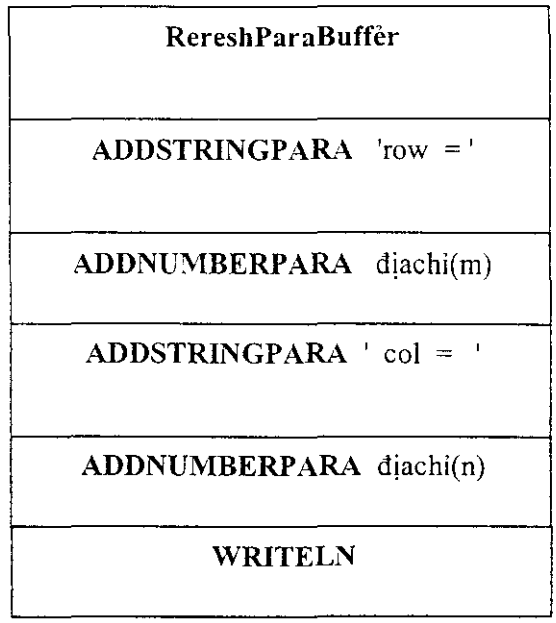
Sơ đồ khối đoạn mã nhị phân giả định như sau:

RefreshParaBuffer
ADDSTRINGPARA 'matmultadd'
ADDNUMBERPARA địachỉ(mt)
ADDNUMBERPARA địachỉ(x)
ADDNUMBERPARA địachỉ(y)
ADDNUMBERPARA địachỉ(w)
EXEC

Hình 4-32 Sơ đồ khối đoạn mã nhị phân giả định câu lệnh exec trong ví dụ 4.8

Ví dụ 4.9. Câu lệnh writeln('row = ' , m , ' col = ' , n)

Sơ đồ khối đoạn mã nhị phân giả định như sau:



Hình 4-33 Sơ đồ khối đoạn mã nhị phân giả định câu lệnh writeln trong ví dụ 4.9.

4.4.4 Thông báo lỗi.

Các lỗi có thể gặp được liệt kê như sau (mỗi lỗi gồm có số nguyên chỉ mã lỗi, tiếp theo là mô tả của lỗi):

1 Unknown Character (ký tự lạ).

2 Name too long (tên quá dài).

3 Logic expression is expected (thiếu toán tử quan hệ).

4 Keyword THEN is expected (thiếu từ khoá THEN).

5 ; after statement is expected (thiếu dấu ; sau câu lệnh).

6 Keyword END is expected (thiếu từ khoá END).

7 Keyword DO is expected (thiếu từ khoá DO).

8 Identifier is expected (thiếu tên).

9 := is expected (thiếu dấu :=).

10 Keyword TO is expected (thiếu từ khoá TO).

11 Keyword BEGIN is expected (thiếu từ khoá BEGIN).

12 . at the end of program is expected (thiếu dấu . kết thúc chương trình).

13 Unknown Type (không hiểu kiểu dữ liệu).

14 ; in variable declaration statement is expected (thiếu dấu ; trong câu lệnh

khai báo biến).

15 : in variable declaration statement is expected (thiếu dấu : trong câu lệnh khai báo biến).

16 Identifier in variable declaration statement is expected (thiếu tên trong phần khai báo biến).

17 Invalid identifier (tên dùng không hợp lệ)

18 Error in Parameters of write statement (Lỗi trong danh sách tham số của câu lệnh hiển thị dữ liệu ra màn hình).

19 String in parameter of select statement is expected (thiếu tham số xâu ký tự trong câu lệnh lấy dữ liệu).

20 Invalid Constant (hằng số không hợp lệ).

21 Unknown identifier (tên chưa được khai báo).

22 Unknown statement (không hiểu câu lệnh).

23) in expression is expected (thiếu dấu ngoặc đóng)

24 (in exec statement is expected (thiếu dấu ngoặc mở trong câu lệnh exec)

25 Function name in exec statement is expected (thiếu tên hàm trong câu lệnh exec).

26 , exec statement is expected (thiếu dấu phảy trong câu lệnh exec).

27) in exec statement is expected (thiếu dấu ngoặc đóng trong câu lệnh exec)

28 (in dbmatgetsize statement is expected (thiếu dấu ngoặc mở trong câu lệnh dbmatgetsize).

29 Constant string parameter in dbmatgetsize statement is expected (thiếu hằng xâu trong câu lệnh dbmatgetsize).

30 , in dbmatgetsize statement is expected (thiếu dấu phảy trong câu lệnh dbmatgetsize).

31 Identifier parameter in dbmatgetsize statement is expected (thiếu tên trong câu lệnh dbmatgetsize).

32) in dbmatgetsize statement is expected (thiếu dấu ngoặc đóng trong câu

lệnh dbmatgetsize);

33 Assignment statement to function is expected (thiếu câu lệnh gán giá trị trả về cho hàm).

34 (in dbvecgetsize statement is expected (thiếu dấu ngoặc mở trong câu lệnh dbvecgetsize).

35 Constant string parameter in dbvecgetsize statement is expected (thiếu hằng xâu trong câu lệnh dbvecgetsize).

36 , in dbvecgetsize statement is expected (thiếu dấu phảy trong câu lệnh dbvecgetsize).

37 Identifier parameter in dbvecgetsize statement is expected (thiếu tên trong câu lệnh dbvecgetsize).

38) in dbvecgetsize statement is expected (thiếu dấu ngoặc đóng trong câu lệnh dbvecgetsize).

39 (in readfromfile statement is expected (thiếu dấu ngoặc mở trong câu lệnh readfromfile).

40 Constant string parameter in readfromfile statement is expected (thiếu hằng xâu trong câu lệnh readfromfile).

41 , in readfromfile statement is expected (thiếu dấu phảy trong câu lệnh readfromfile).

42 Identifier in readfromfile statement is expected (thiếu tên trong câu lệnh readfromfile).

43 Filename in readfromfile statement is expected (thiếu tên file trong câu lệnh readfromfile).

44) in readfromfile statement is expected (thiếu dấu ngoặc đóng trong câu lệnh readfromfile).

45 (in store statement is expected (thiếu dấu ngoặc mở trong câu lệnh store).

46 Constant string in store statement is expected (thiếu hằng xâu trong câu lệnh store).

47 , store statement is expected (thiếu dấu phảy trong câu lệnh store).

48 Identifier in store statement is expected (thiếu tên trong câu lệnh store).

49) in store statement is expected (thiếu dấu ngoặc đóng trong câu lệnh

store).

50] is expected (thiếu dấu]).

51) is expected (thiếu dấu ngoặc mở).

52) in function declaration is expected (thiếu dấu ngoặc đóng trong khai báo chương trình con).

53 (in function declaration is expected (thiếu dấu ngoặc mở trong khai báo chương trình con).

54 Function name in declaration is expected(thiếu tên trong khai báo chương trình con).

55 ; in function declaration is expected (thiếu dấu ; trong khai báo chương trình con).

56 : in function declaration is expected (thiếu dấu : trong khai báo chương trình con).

57 (in writetofile statement is expected (thiếu dấu ngoặc mở trong câu lệnh writetofile).

58 Constant string in writetofile statement is expected (thiếu hằng xâu trong câu lệnh writetofile).

59 , in writetofile statement is expected (thiếu dấu phảy trong câu lệnh writetofile).

60 Identifier in writetofile statement is expected (thiếu tên trong câu lệnh writetofile).

61 Filename in writetofile statement is expected (thiếu tên file trong câu lệnh writetofile).

62) in writetofile statement is expected (thiếu dấu ngoặc đóng trong câu lệnh writetofile).

63 (in dbselectcol statement is expected (thiếu dấu ngoặc mở trong câu lệnh dbselectcol).

64 , in dbselectcol statement is expected (thiếu dấu phảy trong câu lệnh dbselectcol).

65 Identifier in dbselectcol statement is expected (thiếu tên trong câu lệnh dbselectcol).

66) in dbselectcol statement is expected (thiếu dấu ngoặc đóng trong câu lệnh dbselectcol).

67 (in dbselectrow statement is expected (thiếu dấu ngoặc mở trong câu lệnh dbselectrowl).

68 , in dbselectrow statement is expected (thiếu dấu phảy trong câu lệnh dbselectrowl).

69 Identifier in dbselectrow statement is expected (thiếu tên trong câu lệnh dbselectrowl).

70) in dbselectrow statement is expected (thiếu dấu ngoặc đóng trong câu lệnh dbselectrowl).

71 (in dbselectrowcol statement is expected(thiếu dấu ngoặc mở trong câu lệnh dbselectrowlcol).

72 Constant string in dbselectrowcol statement is expected (thiếu hằng xâu trong câu lệnh dbselectrowlcol).

73 , in dbselectrowcol statement is expected(thiếu dấu phảy trong câu lệnh dbselectrowlcol).

74 Identifier in dbselectrowcol statement is expected (thiếu tên trong câu lệnh dbselectrowlcol).

75) in dbselectrowcol statement is expected(thiếu dấu ngoặc đóng trong câu lệnh dbselectrowlcol).

76 (creatematrix statement is expected (thiếu dấu ngoặc mở trong câu lệnh creatematrix).

77 Identifier in creatematrix statement is expected (thiếu tên trong câu lệnh creatematrix).

78 , in creatematrix statement is expected (thiếu dấu phảy trong câu lệnh creatematrix).

79) in creatematrix statement (thiếu dấu ngoặc đóng trong câu lệnh creatematrix).

80 (in createvector statement is expected (thiếu dấu ngoặc mở trong câu lệnh createvector).

81 Identifier in createvector statement is expected (thiếu tên trong câu lệnh createvector).

82 , in createvector statement is expected (thiếu dấu phảy trong câu lệnh createvector).

83) createvector statement is xpected (thiếu dấu ngoặc đóng trong câu lệnh createvector).

84 (in destroymatrix statement is expected (thiếu dấu ngoặc mở trong câu lệnh destroymatrix).

85 Identifier in destroymatrix statement is expected (thiếu tên trong câu lệnh destroymatrix).

86) in destroymatrix statement is expected (thiếu dấu ngoặc đóng trong câu lệnh destroymatrix).

87 (in destroyvector statement is expected (thiếu dấu ngoặc mở trong câu lệnh destroyvector).

88 Identifier in destroyvector statement is expected(thiếu tên trong câu lệnh destroyvector).

89) in destroyvector statement is expected (thiếu dấu ngoặc đóng trong câu lệnh destroyvector).

90 Incompartible type in expression (kiểu không hợp lệnh trong biểu thức).

91 Duplicate identifier (tên khai báo trùng nhau).

92 Unknown this Library Function (câu lệnh trong thư viện tính toán không tồn tại).

93 Parameter incompartible in library function (Tham số không hợp lệ trong câu lệnh gọi thư viện tính toán).

94 Parameter incompartible (tham số không hợp lệ).

95 Constant string is expected (thiếu hằng xâu).

96 (is expected (thiếu dấu ngoặc mở).

97 Error in expression (lỗi trong biểu thức).

98 [is expected (thiếu dấu [ ).

99 Integer Constant is expected (thiếu hằng số nguyên).

100 , is expected (thiếu dấu phảy).

Nếu pha sinh mã không gặp lỗi thì file chương trình mã nhị phân giả định được tạo

ra.

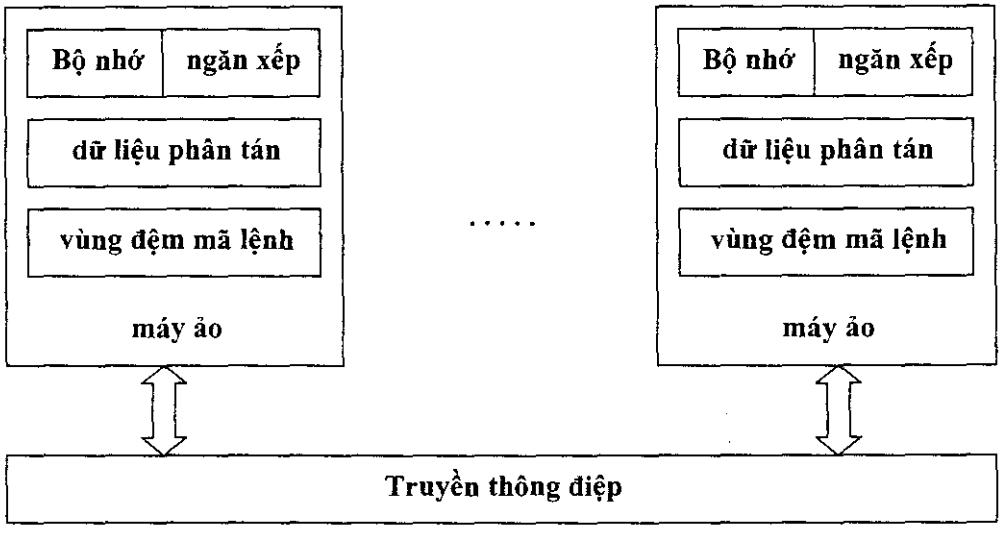
4.5 Áp dụng bộ biên dịch vào hệ thống tính toán

Các chương trình viết bằng ngôn ngữ lập trình nêu trên được bộ biên dịch dịch sang mã nhị phân giả định.

Chương trình mã nhị phân giả định sẽ được máy ảo dịch và thực thi dưới dạng thông dịch. Máy ảo lần lượt đọc từng lệnh, giả mã lệnh và thực hiện lệnh đó.

4.5.1 Mô hình thực thi

Máy ảo thực thi chương trình trong chế độ song song nhiều tiến trình.
Các máy ảo được nhân bản và chạy song song sử dụng thư viện truyền thông điệp MPI.



Hình 4-34 Mô hình thực thi chương trình của máy ảo

Các máy ảo đọc dãy lệnh nhị phân giả định, giải mã lệnh và cho thực hiện. Nội dung bộ nhớ của các biến (trừ biến con trỏ ma trận, vector) và ngăn xếp của các máy ảo luôn được cập nhật giống hệt nhau. Phần dữ liệu của ma trận, vector sẽ được phân tán vào vùng dữ liệu phân tán của từng máy ảo.

Mỗi máy ảo có một định danh riêng.Việc đọc ghi dữ liệu được thực hiện bởi 1 máy ảo có định danh bằng 0. Sau đó dữ liệu này sẽ được truyền đến các máy ảo khác.

4.5.2 Quá trình thực thi

Dãy mã nhị phân giả định của một chương trình nguồn sẽ được đọc vào vùng đệm lệnh của máy ảo. Con trỏ lệnh được trỏ về đầu vùng đệm để đọc lệnh đầu tiên. Chu trình thực hiện chương trình được bắt đầu.

Quá trình thực thi chương trình mã giả định được thực hiện trong chế độ thông dịch, máy ảo lần lượt đọc từng lệnh, giải mã lệnh và thực hiện luôn.Mỗi lệnh giả định có thể coi như là một sự mã hoá của một số lệnh mã máy thực sự. Sự thực hiện của một lệnh tương ứng với lời gọi 1 chương trình con xử lý tương ứng.

Trong mỗi chu trình, máy ảo đọc mã lệnh để xác định chương trình con xử lý tương ứng. Với mỗi mã lệnh, chương trình con tương ứng sẽ đọc các thông tin điều khiển cùng với các toán hạng trong lệnh để thực hiện xử lý dữ liệu. Số lượng toán hạng cũng như kích thước của các lệnh khác nhau tuỳ thuộc voà mã lệnh và thông tin điều khiển. Sau khi thực hiện xong mỗi lệnh, con trỏ lệnh được tự động cập nhật giá trị để trỏ sang lệnh tiếp theo.

Trường hợp mã lệnh là lệnh số học

Máy ảo đọc tiếp các thông tin điều khiển để xác định địa chỉ và kích thước các toán hạng. Đọc tiếp toán hạng, địa chỉ nơi chứa kết quả và thực thi phép tính số học trên các toán hạng đó. Kết quả được lưu vào vùng nhớ trong máy ảo có địa chỉ đã được xác định.

Trường hợp mã lệnh là lệnh so sánh

Tương tự lệnh số học, máy ảo cũng đọc các toán hạng và thực hiện phép so sánh trên hai toán hạng đó.Phép so sánh loại nào trong số 6 loại (> , >= , < , <= , = , <>) được xác định trong mã lệnh. Kết quả của phép so sánh tác động đến giá trị của biến cờ. Nếu kết quả đúng thì biến cờ được thiết lập giá trị bằng 1 và ngược lại thì biến cờ được thiết lập giá trị 0.

Trường hợp mã lệnh là lệnh thao tác ngăn xếp

Nếu mã lệnh xác định là lệnh cất dữ liệu vào ngăn xếp, máy ảo đọc tiếp thông tin điều khiển để xác định địa chỉ và kích thước thực sự của dữ liệu, kích thước mới mà dữ liệu sẽ được chuyển đổi sang trước khi đẩy vào ngăn xếp.Sau đó chuyển đổi dữ liệu sang giá trị tương đương nhưng kích thước mới (ví dụ chuyển dữ liệu là 1 byte sang giá trị tương đương nhưng là số nguyên 4 byte) sau đó đẩy dữ liệu vào ngăn xếp. Con trỏ ngăn xếp được cập nhật giá trị để luôn trỏ lên đỉnh của ngăn xếp.

Nếu dữ liệu cần đưa vào ngăn xếp là một phần tử của ma trận (hoặc vector), máy ảo đọc tiếp các toán hạng để xác định địa chỉ của đối tượng ma trận (vector), chỉ số hàng, cột của phần tử.Từ giá trị địa chỉ ma trận và các chỉ số này, máy ảo sẽ xác định phần tử của ma trận cần lấy giá trị này có thuộc phần dữ liệu phân tán mà nó

quản lý hay không, nếu không phải thì nó sẽ được tiến trình chứa dữ liệu phân tán có mặt phần từ đó gửi đến và giá trị nhận được sẽ được đưa vào ngăn xếp. Nội dung ngăn xếp của tất cả các tiến trình máy ảo đều giống nhau.

Nếu mã lệnh xác định lệnh lấy dữ liệu ra khỏi ngăn xếp, quá trình xử lý diễn ra tương tự như lệnh cất dữ liệu.

Trường hợp mã lệnh là lệnh nhảy

Lệnh nhảy có 3 loại :

- Nhảy không điều kiện
- Nhảy khi giá trị biến cờ bằng 1 (nhảy đúng)
- Nhảy khi giá trị biến cờ bằng 0 (nhảy sai)

Máy ảo đọc tiếp thông tin điều khiển để xác định địa chỉ nơi chứa giá trị của nhãn sau đó đọc giá trị của nhãn.Giá trị này chính là địa chỉ của lệnh sẽ được con trỏ lệnh nhảy tới.

- Nếu là lệnh nhảy không điều kiện thì cập nhật giá trị của nhãn cho con trỏ lệnh.
- Nếu là lệnh nhảy đúng thì chỉ cập nhật giá trị mới cho con trỏ lệnh nếu giá trị biến cờ bằng 1.
- Nếu là lệnh nhảy sai thì chỉ cập nhật giá trị mới cho con trỏ lệnh nếu giá trị biến cờ bằng 0.

Trường hợp mã lệnh xác định lệnh làm tươi vùng đệm truyền tham số

Máy ảo sẽ thiết lập giá trị đưa con trỏ của vùng đệm trỏ về đầu vùng đệm. Mỗi khi có lệnh đưa dữ liệu ra vùng đệm thì con trỏ vùng đệm lại được tăng lên để trỏ về cuối phần dữ liệu có giá trị trong vùng đệm.

Trường hợp mã lệnh xác định lệnh đưa giá trị số ra vùng đệm truyền tham số

Máy ảo đọc thông tin điều khiển để xác định địa chỉ và kích thước dữ liệu cần đưa ra vùng đệm, sau đó thực hiện đọc dữ liệu và đưa ra vùng đệm.

Trường hợp mã lệnh xác định lệnh đưa xâu ký tự ra vùng đệm truyền tham số

Máy ảo đọc thông tin điều khiển để xác định độ dài xâu ký tự cần đọc, sau đó đọc xâu ký tự trong với độ dài đã được xác định và đưa xâu ký tự đó ra vùng đệm.

Trường hợp mã lệnh là lệnh đọc ghi file

Con trỏ vùng đệm được thiết lập về 0 để trỏ vào đầu vùng đệm.

Mỗi tham số đều có hai byte điều khiển xác định loại dữ liệu (xâu, số) và độ dài của dữ liệu trong vùng đệm.Các thông tin này được xác định trong các câu lệnh đưa

tham số ra vùng đệm.

- Máy ảo đọc tham số đầu tiên trong vùng đệm, tham số này là xâu ký tự xác định đối tượng làm việc là ma trận hay vector. Con trỏ vùng đệm được cập nhật giá trị để trỏ vào đầu tham số tiếp theo.
- Máy ảo đọc tiếp tham số thứ hai. Đây là giá trị địa chỉ của ngăn nhớ 4 byte trong bộ nhớ của máy ảo nơi chứa địa chỉ thực sự của ma trận (vector). Con trỏ vùng đệm được cập nhật giá trị để trỏ vào tham số tiếp theo.
- Máy ảo đọc tham số thứ 3, tham số này là một xâu xác định tên file sẽ được đọc hoặc ghi.

Giả sử là lệnh ghi dữ liệu ra file.Sau khi có đầy đủ tất cả các tham số cần thiết , máy ảo có định danh bằng 0 sẽ thực hiện việc đọc ghi file. Nếu dữ liệu của ma trận (vector) không nằm trong phần dữ liệu phân tán của tiến trình 0 thì nó sẽ nhận được dữ liệu từ các tiến trình khác gửi đến và ghi ra file.

Nếu mã lệnh xác định lệnh đọc dữ liệu (giả sử ma trận) từ file thì máy ảo sẽ mở file và đọc kích thước ma trận, sau đó gọi hàm khởi tạo ma trận với kích thước đã được xác định. Khi đó các khối dữ liệu của ma trận sẽ được phân tán trên các tiến trình, giá trị của các khối này chưa được xác định. Sau đó tiến trình có định danh bằng 0 sẽ thực hiện đọc dữ liệu lần lượt từ file. Tiến trình này sẽ truyền giá trị của phần tử đọc được sang đúng tiến trình sở hữu khối dữ liệu phân tán mà nó thuộc vào và giá trị đó sẽ được cập nhật vào đùng vị trí.

Trường hợp mã lệnh là lệnh gọi thư viện tính toán

Tương tự lệnh đọc ghi file, máy ảo đọc tham số đầu tiên là một xâu ký tự từ vùng đệm truyền tham số, xâu này xác định tên hàm trong thư viện sẽ được gọi thực hiện. Tiếp đó dựa vào giá trị của tên hàm, máy ảo đọc các tham số tiếp theo xác định địa chỉ của các đối tượng dữ liệu ma trận,vector là tham số trong lời gọi hàm và cuối cùng là thực hiện gọi hàm thư viện.Địa chỉ của tham số ma trận(vector) là kết quả trong lời gọi hàm sẽ được cập nhật vào bộ nhớ của máy ảo.

Trường hợp mã lệnh là lệnh truy xuất dữ liệu trong cơ sở dữ liệu

Tương tự lệnh gọi thư viện tính toán, các tham số trong các hàm truy xuất dữ liệu được lấy từ vùng đệm truyền tham số.

Con trỏ vùng đệm được thiết lập về 0.

Máy ảo đọc các tham số từ vùng đệm như xâu ký tự xác định đối tượng là ma trận hay vector, xâu ký tự xác định tên của đối tượng ma trận(vector) trong cơ sở dữ liệu, giá trị các chỉ số. Sau đó thực hiện lời gọi hàm truy xuất dữ liệu.

Trường hợp mã lệnh xác định lệnh hiển thị dữ liệu ra màn hình

Cũng như các lệnh trên, dữ liệu cần hiển thị là xâu , số đã được đưa ra vùng đệm truyền tham số. Con trỏ vùng đệm được thiết lập về 0.

Tiếp đó máy ảo lần lượt đọc các giá trị tham số từ vùng đệm và đưa ra màn hình.

4.5.3 Xử lý lỗi thực thi.

Trong quá trình thực thi cũng rất hay gặp lỗi, các lỗi này không thể phát hiện được trong các pha của quá trình biên dịch, các lỗi có thể gặp đó là:

- Lỗi khi chi cho 0.
- Truy xuất dữ liệu của ma trận, vector chưa được khởi tạo.
- Kích thước của ma trận, vector không hợp lệ trong các phép tính toán.
- Tràn bộ nhớ, ngăn xếp, vùng đệm truyền tham số.

Khi gặp một trong các lỗi thực thi này, máy ảo dừng việc thực thi và thông báo lỗi.

4.6 Kết quả đạt được và hướng phát triển

4.6.1 Kết quả đạt được

Bộ biên dịch được xây dựng để đáp ứng mục tiêu của hệ thống là cung cấp cho người một ngôn ngữ lập trình đơn giản, có tích hợp các câu lệnh truy xuất dữ liệu và gọi thư viện tính toán đã được xây dựng sẵn. Người dùng lập trình bằng ngôn ngữ này không cần thiết phải hiểu về lập trình song song, họ chỉ cần biết trong hệ thống đã có những thư viện tính toán nào, từ đó người lập trình có thể tổ hợp các thư viện tính toán trong hệ thống để tạo ra chương trình giải quyết bài toán đặt ra. Dữ liệu dùng cho việc tính toán có thể lấy từ cơ sở dữ liệu chuyên dụng của hệ thống bằng các câu lệnh truy xuất dữ liệu, có thể đọc từ file. Kết quả tính toán có thể được lưu vào cơ sở dữ liệu, file hoặc hiển thị ngay ra màn hình.

Bộ biên dịch có nhiệm vụ dịch chương trình một dạng mã nhị phân giả định. Mã nhị phân giả định là một dạng mã trung gian tại đó các tên biến được ánh xạ hết sang địa chỉ, mỗi câu lệnh trong chương trình nguồn được phân tách thành các câu lệnh giả định nhỏ hơn.

Vì thời gian có hạn nên còn một số mục tiêu của hệ thống đặt ra chưa được hoàn thiện, hiệu năng thực thi chương trình chưa được cao. Sau đây là một số tổng kết về kết quả đạt được và hướng phát triển tiếp theo.

Về cơ bản, các kết quả đã đạt được là

- Xây dựng xong bộ biên dịch bằng ngôn ngữ Delphi chạy trên hệ điều hành Linux.

- Xây dựng xong máy ảo.

4.6.2 Định hướng phát triển

- Bổ sung các kiểu dữ liệu mới.
- Tối ưu mã, cải tiến máy ảo để nâng cao hiệu năng thực thi chương trình.

4.7 Phụ lục chương 4

4.7.1 Các module chính trong chương trình

Bộ trình dịch được tổ chức thành các Module sau:

- Module định nghĩa các cấu trúc dữ liệu `PcsDataDefine`. Module này định nghĩa các cấu trúc dữ liệu được sử dụng trong chương trình.
- Module phân tích từ vựng `PcsLexemelParser`. Module này có hàm chính là

```
function LexemeParse(src:string;Var err:ErrorType):TList;
```

Hàm này trả về danh sách các từ vựng được tách ra từ chương trình nguồn, biến err dùng để chứa các thông tin về lỗi nếu có.

- Module phân tích cú pháp PcsSyntaxParser .Module này chứa các thủ tục phục vụ cho việc phân tích cú pháp. Các chương trình con phân tích cú pháp theo phương pháp đệ quy trên xuống được liệt kê theo theo tên (chỉ có phần đầu của thủ tục đó) như sau:

```
procedure  PROG(p : pItem);
procedure  VAR_DECL_LIST(p : pItem);
procedure  FUNC_DECL_LIST(p : pItem);
procedure  FUNC_DECL_ITEM(p : pItem);
procedure  LIST_STMT(p : pItem);
procedure  VAR_DECL_ITEM( p : pItem);
procedure  VAR_LIST(p : pItem);
procedure  VAR_TYPE(p : pItem);
procedure  ARGUMENTS(p : pItem);
procedure  EXPRESSION_LIST(p : pItem);
procedure  STATEMENT(p : pItem);
procedure  ASSIGN_STMT(p : pItem);
procedure  LOOP_STMT(p : pItem);
```

```
procedure  CONDITION_STMT(p : pItem);
procedure  OUTPUT_STMT(p : pItem);
procedure  OUTPUT_TOFILE_STMT(p : pItem);
procedure  OUTPUT_TOSCREEN_STMT(p : pItem);
procedure  INPUT_STMT(p : pItem);
procedure  VARIABLE(p : pItem);
procedure  EXPRESSION(p : pItem);
procedure  TERM(p : pItem);
procedure  FACTOR(p : pItem);
procedure  FACTOR_ITEM(p : pItem);
procedure  CONDITION(p : pItem);
procedure  CONDITION_TERM(p : pItem);
procedure  CONDITION_FACTOR(p : pItem);
procedure  CREATE_MATRIX_STMT(p : pItem);
procedure  CREATE_VECTOR_STMT(p : pItem);
procedure  DESTROY_MATRIX_STMT(p : pItem);
procedure  DESTROY_VECTOR_STMT(p : pItem);
procedure  SELECT_ROWCOL_STMT(p : pItem);
procedure  SELECT_ROW_STMT(p : pItem);
procedure  SELECT_COL_STMT(p : pItem);
procedure  STORE_STMT(p : pItem);
procedure  MAT_GET_SIZE_STMT(p : pItem);
procedure  VEC_GET_SIZE_STMT(p : pItem);
procedure  SELECT_PARA(p : pItem);
procedure  TWO_VALUE(p : pItem);
procedure  LIB_STMT(p : pItem);
procedure  DB_STMT(p : pItem);
procedure  EXEC_PARA_LIST(p : pItem);
procedure  EXEC_PARA_ITEM(p : pItem);
procedure  OUTPUT_DATA_LIST( p : pItem);
procedure  OUTPUT_DATA_ITEM( p : pItem);
procedure GetSym;
procedure Error(Line : integer; errdes : string);
procedure Main;
```

Ngoài ra còn có các thủ tục phụ khác thực hiện chuẩn hoá cây cú pháp.

- Module sinh mã PcsCodeGenerator. Module này chứa các thủ tục phục vụ cho việc sinh mã nhị phân giả định. Các hàm chính như sau:

```
function GenerateStatement(node:pItem;var lstVar:TList):ErrorType;
function GenerateListStatement(node:pItem;
                                    var lstVar:TList):ErrorType;
function GenerateAssignStmtCode(node:pItem;_PC:integer;
                                    var lstVar:TList):ErrorType;
function GenerateAssignment_EXPR_TO_VARIABLE(node:pItem;
                                    var lstVar:TList):ErrorType;
function Generate_Expression_Value_Code(node:pItem;
                                    var lstVar: TList):ErrorType;
function GenerateOutPutStmt(node:pItem;var lstVar:TList): ErrorType;
function Generate_READ_FROM_FILE_Stmt(node:pItem;
                                        var lstVar:TList):ErrorType;
function Generate_WRITE_TO_FILE_Stmt(node:pItem;
                                        var lstVar:TList):ErrorType;
function GenerateComparasionStmt(node:pItem;
                                    var lstVar:TList):ErrorType;
function GenerateLogicExpression(node:pItem;
                                    var lstVar:TList):ErrorType;
function Generate_IF_THEN_Stmt(node:pItem;
                                    var lstVar:TList):ErrorType;
function Generate_IF_THEN_ELSE_Stmt(node:pItem;
                                    var lstVar:TList):ErrorType;
function Generate_WHILE_DO_Stmt(node:pItem;
                                    var lstVar:TList):ErrorType;
function Generate_FOR_Stmt(node:pItem):ErrorType;
function Generate_ADD_NUMBER_PARA_ToList(node:pItem;
                                    var lstVar:TList):ErrorType;
function Generate_ADD_ADDR_PARA_ToList(pv:pVariable):ErrorType;
function Generate_ADD_STRING_PARA_ToList(node:pItem):ErrorType;
function Generate_PUT_PARA_LIST_ToBuffer(node:pItem;
                        var lstVar:TList;Ext:integer=0):ErrorType;
function Generate_CREATE_STMT(node:pItem;AddrRs:integer;
                        var lstVar:TList):ErrorType;
```

```
function Generate_MAT_GET_SIZE_STMT(node:pItem;AddrRs:integer;
                    var lstVar:TList):ErrorType;
function Generate_VEC_GET_SIZE_STMT(node:pItem;AddrRs:integer;
                    var lstVar:TList):ErrorType;
function Generate_SELECT_ROW_COL_STMT(node:pItem;AddrRs:integer;
                    var lstVar:TList):ErrorType;
function Generate_SELECT_ROWCOL_STMT(node:pItem;AddrRs:integer;
                    var lstVar:TList):ErrorType;
function Generate_STORE_STMT(node:pItem;AddrRs:integer;
                    var lstVar:TList):ErrorType;
function Generate_EXEC_STMT(node:pItem;AddrRs:integer;
                    var lstVar:TList):ErrorType;
```

Ngoài ra còn có các thủ tục phụ khác có chức năng:

- Thao tác với bảng ký hiệu.
- Kiểm tra sự phù hợp về kiểu dữ liệu trong các câu lệnh.

4.7.2 Thực hiện biên dịch và chạy chương trình

Yêu cầu hệ thống :

- Đã cài đặt thư viện truyền thông LAM/MPI
- Đã có thư viện PETSC được cấu hình để làm việc trên nhiều node dựa trên thư viện truyền thông LAM/MPI.

Mô tả thư mục

- Các bước dưới đây được thực hiện với source của chương trình nằm trong thư mục /home/k43/sannh/pcsroot/
- Đặt đường dẫn PCSROOT= /home/k43/pcsroot
- Thư mục $PCSROOT/home/sannh/src chứa các file chương trình nguồn viết bởi ngôn ngữ giả pascal.
- $PCSROOT/home/sannh/bin : chứa các file dạng mã nhị phân giả định sau biên dịch.

Dịch chương trình

- Bước 1: Dịch chương trình nguồn ra mã giả định.

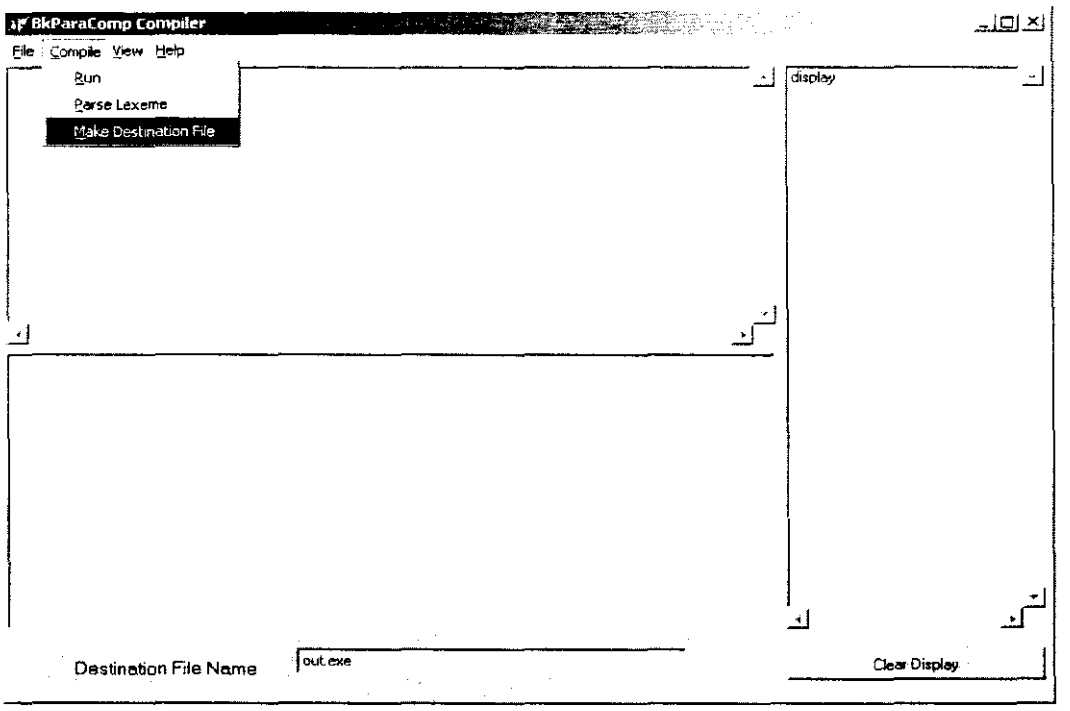
```
$./pcsc $PCSROOT/home/sannh/src/test6.pcs
$PCSROOT/home/sannh/bin/test6
```

- Bước 2: Thực hiện chạy máy ảo .

```
$mpirun -np 4  pcsvm   test6
```

4.7.3 Giao diện chương trình và hướng dẫn sử dụng

Giao diện chương trình



Hình 4-35 Giao diện của chương trình.

Hướng dẫn sử dụng chương trình.

* Soạn thảo chương trình nguồn trong ô soạn thảo ở góc trên bên trái bằng 1 trong 2 cách sau:

- Soạn thảo trực tiếp.
- Mở chương trình đã có từ file bằng cách chọn chức năng Open trong mục File.

* Để thực hiện chức năng biên dịch chương trình, chọn chức năng Make Destination File trong mục Compile.

* Để thực hiện phân tích từ vựng, chọn chức năng Parse Lexeme trong mục Compile.

* Để chạy chương trình, chọn chức năng Run trong mục Compile.

* Các chức năng khác:

- Chức năng View Destination File mục View dùng để hiển thị file nhị phân giả định.
- Chức năng New trong mục File dùng để mở soạn thảo chương trình mới.

- Chức năng Save trong mục File dùng để lưu chương trình nguồn đang soạn thảo ra file.
- Chức năng Use guide trong mục Help là hướng dẫn người sử dụng.

4.7.4 Một số kết quả thử nghiệm

Sau đây là một số kết quả thử nghiệm, trong đó có chương trình nguồn và chương trình đích tương ứng.

Chương trình đích là chương trình mã nhị phân giả định (các byte được biểu diễn dưới dạng Hexa), bên cạnh mỗi lệnh nhị phân giả định có các lời giải thích trong dấu ngoặc.

2.3.1. Chương trình 1

Chương trình này đưa ra ví dụ về câu lệnh gán.

Chương trình nguồn

```
var n,m : integer;
begin
      m := 10;
      n := 123*m;
      writeln(n);
end.
```

Chương trình đích

```
0C 00 44 07 00 00 00   (Lenh JMP, nhảy đến vị trí 7)
60 01 14 0A 32 00 00 00   (Lenh Move gía trị 10 vào ngăn nhớ tại địa chỉ 50 tức là biến m)
0F 00 44 7B 00 00 00   (Lenh PUSH, cất giá trị 123 vào ngăn xếp)
60 11 44 32 00 00 00 12 00 00 00   (Lenh Move, gán giá trị biến tại địa chỉ 50 tức là biến m cho biến trung gian tại địa chỉ 18)
10 10 44 03 00 00 00   (Lenh POP, lấy giá trị 123 ra khỏi ngăn xếp và đưa vào biến trung gian tại địa chỉ 3)
02 30 44 03 00 00 00 12 00 00 00 2E 00 00 00   (Lenh MULT, thực hiện nhân giá trị của biến tại địa chỉ 3 và biến tại địa chỉ 18 và đưa kết quả vào biến tại địa chỉ 46 tức là biến n)
24 00 01   (Lenh RefreshParaBuffer)
```

```
0F 10 44 2E 00 00 00    (Lenh PUSH, cất giá trị biến tại địa chỉ
46 tức là biến n vào ngăn xếp)

10 10 44 03 00 00 00    (Lenh POP, lấy giá trị ra khỏi ngăn xếp và
đưa vào biến trung gian tại địa chỉ 3)

22 10 44 03 00 00 00    (Lenh AddNumberPara, đưa giá trị của biến
trung gian tại địa chỉ 3 ra vùng đệm)

26 00 00    (Lenh WRITELN, hiển thị giá trị ở vùng đệm ra màn
hình)
```

Kết quả thực hiện chương trình

màn hình hiển thị:

```
1230
```

2.3.2. Chương trình 2

Chương trình này đưa ra ví dụ về câu lệnh gọi thư viện tính toán, các câu lệnh vào ra qua file.

Chương trình nguồn

```
var
A : matrix[5,5];
x,b : vector[5];
i,j,size : integer;
begin
   readfromfile('matrix',A,'mt3.txt');
   readfromfile('vector',b,'v1.txt');
   exec('vecgetsize',b,size);
   createvector(x,size);
   exec('sles',A,x,b);
   for i := 0 to size - 1 do write(x[i],'  ');
end.
```

Chương trình đích

```
0C 00 44 07 00 00 00   (Lenh JMP, nhảy đến địa chỉ 7)

24 00 03   (Lenh RefreshParaBuffer)

23 00 06 6D 61 74 72 69 78   (Lenh AddStringPara, đưa xâu
ký tự 'matrix' ra vùng đệm)

22 00 44 2E 00 00 00   (Lenh AddNumberPara, đưa giá trị 46
tức là địa chỉ của biến A ra vùng đệm)

23 00 07 4D 54 41 2E 54 58 54   (Lenh AddStringPara. đưa
xâu ký tự 'MTA.TXT' ra vùng đệm)

11 00 00   (Lenh ReadFromFile, gọi lệnh readfromfile với
các tham số lấy từ vùng đệm)

24 00 03   (Lenh RefreshParaBuffer)

23 00 06 76 65 63 74 6F 72   (Lenh AddStringPara, đưa xâu
ký tự 'vector' ra vùng đệm)

22 00 44 36 00 00 00   (Lenh AddNumberPara, đưa giá trị 54
tức là địa chỉ của biến b ra vùng đệm)

23 00 0B 56 45 43 54 4F 52 42 2E 54 58 54   (Lenh
AddStringPara, đưa xâu ký tự 'VECTORB.TXT' ra vùng đệm)

11 00 00   (Lenh ReadFromFile, thực hiện lệnh readfromfile
với các tham số lấy từ vùng dệm)

24 00 03   (Lenh RefreshParaBuffer)

23 00 0A 76 65 63 67 65 74 73 69 7A 65   (Lenh
AddStringPara, đưa xâu ký tự 'vecgetsize' ra vùng đệm)

22 00 44 36 00 00 00   (Lenh AddNumberPara, đưa giá trị 54
tức địa chỉ của biến b ra vùng đệm)

22 00 44 3A 00 00 00   (Lenh AddNumberPara, đưa giá trị 58
tức địa chỉ của biến size ra vùng đệm)

50 00 00   (Lenh Exec, thực hiện lời gọi thư viện tính toán
với hàn vecgetsize để lấy kích thước của biến vector b đưa vào
biến size)

24 00 00   (Lenh RefreshParaBuffer)

22 00 44 32 00 00 00   (Lenh AddNumberPara, đưa giá trị 50
tức địa chỉ của biến x ra vùng đệm)

0F 10 44 3A 00 00 00   (Lenh PUSH, đưa giá trị 58, địa chỉ
của biến size vào ngăn xếp)

10 10 44 03 00 00 00   (Lenh POP, lấy giá trị ra khỏi ngăn
xếp và đưa vào biến trung gian tại địa chỉ 3)

22 10 44 03 00 00 00   (Lenh AddNumberPara, đưa giá trị của
```

```
biến trung gian tại địa chỉ 3 ra vùng đệm)

     21 00 00    (Lenh CreateVector, thực hiện lệnh createvector
với các tham số lấy từ vùng đệm)

     24 00 04    (Lenh RefreshParaBuffer)

     23 00 04 73 6C 65 73    (Lenh AddStringPara, đưa xâu ký tự
'sles' ra vùng đệm)

     22 00 44 2E 00 00 00    (Lenh AddNumberPara, đưa giá trị 46
tức địa chỉ biến A ra vùng đệm)

     22 00 44 32 00 00 00    (Lenh AddNumberPara, đưa giá trị 50
tức địa chỉ biến x ra vùng đệm)

     22 00 44 36 00 00 00    (Lenh AddNumberPara, đưa giá trị 58
tức địa chỉ biến b ra vùng đệm)

     50 00 00    (Lenh Exec, thực hiện gọi thư viện tính toán
sles để giải hệ tuyến tính Ax = b)

60 01 14 00 3A 00 00 00    (Lenh Move giá trị 0 vào ô nhớ tại địa
chỉ 58 tức biến i)

0F 10 44 3A 00 00 00    (Lenh PUSH cất giá trị biến i vào ngăn
xếp)

0F 10 44 42 00 00 00    (Lenh PUSH cất giá trị biến size vào ngăn
xếp)

60 01 14 01 12 00 00 00    (Lenh Move giá trị 1 vào biến trung
gian tại địa chỉ 18)

10 10 44 03 00 00 00    (Lenh POP lấy giá trị ra khỏi ngăn xếp tức
giá trị biến size và đưa vào biến trung gian tại địa chỉ 3)

01 30 44 03 00 00 00 12 00 00 00 21 00 00 00    (Lenh SUB thực
hiện trừ nội dung biến tại địa chỉ 3 tức biến size cho biến tại
địa chỉ 18 và kết quả đưa vào biến trung gian tại địa chỉ 33)

0F 10 44 21 00 00 00    (Lenh PUSH cất giá trị của biến trung gian
tại địa chỉ 33 vào ngăn xếp)

10 10 84 16 00 00 00    (Lenh POP lấy giá trị ra khỏi ngăn xếp,
chuyển thành kiểu double và đưa giá trị vào biến trung gian tại
địa chỉ 22)

10 10 84 07 00 00 00    (Lenh POP lấy giá trị ra khỏi ngăn xếp từ
giá trị biến i và chuyển thành kiểu double và đưa vào biến trung
gian tại địa chỉ 7)

09 38 00 07 00 00 00 16 00 00 00    (Lenh Leq thực hiện phép so
sánh < nội dung của biến trung gian tại địa chỉ 7 và địa chỉ 22)

0E 00 00 6D 01 00 00    (Lenh JF thực hiện nhảy ra khỏi chương
trình nếu cờ bằng 0)
```

```
24 00 02    (Lenh RefreshParaBuffer làm tươi vùng đệm truyền tham
số)

60 11 44 3A 00 00 00 03 00 00 00    (Lenh Move giá trị ô nhớ có
địa chỉ 58 tức biến i  vào biến trung gian tại địa chỉ 3)

0F 30 88 32 00 00 00 03 00 00 00    (Lenh PUSH cất giá trị của
x[i] vào ngăn xếp)

10 10 88 07 00 00 00    (Lenh POP lấy giá trị của x[i] ra khỏi
ngăn xếp và đưa vào biến trung gian tại địa chỉ 7).

22 10 88 07 00 00 00    (Lenh AddNumberPara đưa giá trị của biến
trung gian tại địa chỉ 7 ra vùng đệm)

23 00 02 20 20    (Lenh AddStringPara đưa xâu ký tự '  ' ra vùng
đệm)

25 00 00    (Lenh WRITE thực hiện ghi các dữ liệu ở vùng đệm ra
màn hình)

0F 10 44 3A 00 00 00    (Lenh PUSH cất giá trị biến i vào ngăn
xếp)

60 01 14 01 12 00 00 00    (Lenh Move giá trị 1 vào biến trung
gian tại địa chỉ 18)

10 10 44 03 00 00 00    (Lenh POP lấy giá trị biến i ra khỏi ngăn
xếp và đưa vào biến trung gian tại địa chỉ 3)

00 30 44 03 00 00 00 12 00 00 00 3A 00 00 00    (Lenh ADD cộng i
với 1 và kết quả đưa vào i)

0C 00 00 BF 00 00 00    (Lenh JMP nhảy lên nhãn của vòng lặp )□
```

Kết quả thực hiện chương trình

Trong chương trình trên dữ liệu cho ma trận A được đọc từ file mt3.txt có nội dung như sau:

```
5 5

1 4 1 2 1

2 1 2 3 2

0 0 1 3 0

0 0 0 2 1

1 1 1 1 1
```

Dữ liệu vector b được đọc từ file v1.txt có nội dung như sau:

```
5
2 1 1 2 0
```

Màn hình hiển thị kết quả:

```
1.750000  0.250000  -2.750000  1.250000  -0.500000
```

2.3.3. Chương trình 3

Chương trình này đưa ra ví dụ minh họa câu lệnh rẽ nhánh.

Chương trình nguồn

```
var
a : integer;
begin
   a := 32;
   if a mod 2 = 0 then write('a chan')
   else write('a le');
end.
```

Chương trình đích

```
0C 00 44 07 00 00 00    (Lenh JMP, nhảy đến địa chỉ 7 tức thân chương trình chính)

60 01 14 20 2E 00 00 00   (Lenh Move, gán giá trị 32 cho biến tại địa chỉ 46 tức là biến a)

0F 10 44 2E 00 00 00    (Lenh PUSH, đưa giá trị biến tại địa chỉ 46 vào ngăn xếp)

60 01 14 02 12 00 00 00    (Lenh Move, gán giá trị 2 cho biến trung gian tại địa chỉ 18)

10 10 44 03 00 00 00    (Lenh POP, lấy giá trị ra khỏi ngăn xếp tức giá trị của biến a và đưa  vào biến trung gian tại địa chỉ 3)

05 30 44 03 00 00 00 12 00 00 00 21 00 00 00    (Lenh MOD, thực hiện phép toán số học MOD, lấy giá trị của biến tại địa chỉ 3 chia lấy dư cho biến tại địa chỉ 18 và kết quả đưa vào biến trung gian tại địa chỉ 33)
```

```
0F 10 44 21 00 00 00    (Lenh PUSH, đưa giá trị của biến tại địa chỉ 33 vào ngăn xếp)

0F 00 44 00 00 00 00    (Lenh PUSH, đưa giá trị 0 vào ngăn xếp)

10 10 84 16 00 00 00    (Lenh POP, lấy giá trị ra khỏi ngăn xếp tức giá trị 0 và đưa vào biến trung gian tại địa chỉ 22)

10 10 84 07 00 00 00    (Lenh POP, lấy giá trị ra khỏi ngăn xếp tức giá trị của biến trung gian tại địa chỉ 33 và đưa vào biến trung gian tại địa chỉ 7)

0A 38 00 07 00 00 00 16 00 00 00    (Lenh Equ,  thực hiện phép so sánh = trên biến trung gian tại địa chỉ 7 và biến trung gian tại địa chỉ 22)

0E 00 00 78 00 00 00    (Lenh JF, thực hiện nhảy đến địa chỉ 120 của vùng đệm lệnh nếu kết quả so sánh sai)

24 00 01    (Lenh RefreshParaBuffer)

23 00 06 61 20 63 68 61 6E    (Lenh AddStringPara, đưa xâu ký tự 'a chan' ra vùng đệm)

25 00 00    (Lenh WRITE, thực hiện hiển lệnh thị xâu ký tự từ vùng đệm ra màn hình)

0C 00 00 85 00 00 00    (Lenh JMP, nhảy đến địa chỉ 133 trong vùng đệm lênh)

24 00 01    (Lenh RefreshParaBuffer)

23 00 04 61 20 6C 65    (Lenh AddStringPara, đưa xâu ký tự 'a le' ra vùng đệm)

25 00 00    (Lenh WRITE, thực hiện lệnh hiển thị  xâu ký tự ở vùng đệm ra màn hình)
```

Kết quả thực hiện chương trình

Màn hình hiển thị kết quả:

```
a chan
```

CHƯƠNG 5 Hệ Thống Thực Thi Tính Toán

5.1 Đặc tả và phân tích thiết kế hệ thống

5.1.1 Đặc tả hệ thống

Như đã nói ở phần trước, PBS là một bộ lập lich hỗ trợ rất tốt cho môi trường tính toán song song. Tuy nhiên một nhược điểm của bộ lập lịch này là rất khó sử dụng, đặc biệt là với những người sử dụng không chuyên. Mỗi câu lệnh của PBS cung cấp cho người sử dụng có thể có đến hàng chục tham số mà đôi khi những tham số này không có ý nghĩa gì đối với người sử dụng. Vậy nhiệm vụ của bộ công cụ hỗ trợ người dùng do nhóm BKluster đưa ra là làm đơn giản các thao tác của PBS mà vẫn đảm bảo đầy đủ chức năng cung cấp cho người sử dụng. Ngoài ra phải xây dựng một giao diện cho chương trình qua đó cho phép người dùng có thể tương tác trong môi trường đồ họa nhằm tạo điều kiện thuận lợi nhất cho người sử dụng.

Bộ công cụ User Tool phép người dùng có thể tương tác với hệ thống BKluster, qua đấy người dùng có thể tiến hành chạy các công việc song song cũng như theo dõi các công việc song song mà mình đã đệ trình với hệ thống.

Bên cạnh các yếu tố trên thì một yêu cầu đối với bộ công cụ User Tool là phải có khả năng độc lập với hệ điều hành để có thể cho phép người dùng cho dù sử dụng bất kể hệ điều hành nào cũng có thể tương tác một cách thoải mái với hệ thống. Đây là một yêu cầu tương đối phức tạp vì như chúng ta đều biết thì PBS là một dự án nguồn mở nên rất dễ dàng trong việc tích hợp nó dưới nền hệ điều hành Linux nhưng để có thể sử dụng trong hệ điều hành windows thì lại là cả một vấn đề, trong khi đó thì ở nước ta nhìn chung mọi người vẫn chỉ quen với hệ điều hành windows còn hệ điều hành Linux vẫn còn xa lạ với người sử dụng. Sau một thời gian tìm hiểu, một giải pháp cho vấn đề này là xây dựng hệ thống theo mô hình Client-Server, trong đó Server là một ứng dụng chạy trên một máy chủ dưới hệ điều hành Linux dùng có cài đặt PBS ở dưới, còn Client sẽ là phần giao tiếp với người sử dụng sẽ được đặt tại các máy của người sử dụng có khả năng chạy độc lập với hệ điều hành và giao tiếp với Server thông qua Socket. Chức năng cụ thể của từng phần sẽ được trình bày kỹ hơn trong phần kiến trúc hệ thống.

Để có thể tiến hành triển khai thực hiện cài đặt bộ User Tool đầu tiên chúng ta phải xác định rõ bộ công cụ sẽ hỗ trợ cho người dùng những chức năng nào và yêu cầu cụ thể của từng chức năng sẽ như thế nào.

Đầu tiên chúng ta cần biết hệ thống BKluster sẽ có những người dùng như thế nào và cụ thể hệ thống của chúng ta sẽ hỗ trợ cho người dùng nào trong đó nữa. Theo thiết kế ban đầu thì hệ thống sẽ chia làm ba loại người dùng với các quyền hạn cụ

thể như sau:

- Đầu tiên là những người dùng có quyền thấp nhất của hệ thống. Đây là những người dùng chỉ có quyền xem qua một cách tổng quan các tài nguyên tính toán của hệ thống để biết một cach sơ qua về sức mạnh của hệ thống.
- Lớp người dùng thứ hai là lớp người dùng có nhiệm vụ làm công việc quản trị hệ thống. Đây là những người dùng có quyền cao nhất trong hệ thống. Những người dùng này có thể cấu hình lại toàn bộ hệ thống, đảm bảo cho toàn bộ hệ thống chạy một cách hoàn hảo và tiến hành các thao tác quản lý người dùng. Các công cụ hỗ trợ cho lớp người dùng này sẽ nằm trong bộ Admin Tool.
- Cuối cùng là lớp có quyền thực thi các thao tác với các job. Đây chính là đối tượng tương tác chính với bộ công cụ User Tool mà em xây dựng. Những người dùng này cần được người quản trị cấp cho một account và một thư mục riêng gọi là thư mục làm việc. Sau khi được cấp một account thì mỗi lần muốn thao tác với hệ thống thì họ sẽ đăng nhập vào hệ thống với account này và thực thi các công việc mà mình muốn thực hiện, mọi kết quả thao tác của họ sẽ được lưu lại ngay trên thư mục riêng của họ đã được người quản trị cung cấp cho.

Trên đây chúng ta đã thấy một cách tổng quan về người dùng của hệ thống BKluster. Để phục vụ tốt cho việc phân tích thiết kế và tiến đến cài đặt bộ công cụ User Tool chúng ta sẽ cùng tìm hiểu kỹ về lớp người dùng thứ ba và các công việc mà những người dùng này được phép thực hiện.

Như đã nói ở trên đây chính là lớp người dùng thao tác chính với bộ công cụ User Tool. Bộ công cụ được xây dựng sẽ hỗ trợ cho người dùng có khả năng giao tiếp một cách dễ dàng với PBS, nghĩa là PBS cung cấp lệnh gì cho những người dùng thông thường thì bộ User Tool cũng phải cung cấp đầy đủ cho người sử dụng nhưng chi khác với PBS là giao diên của BKluster nói chùng và User Tool nói riêng sẽ dễ hiểu và dễ sử dụng hơn. Như vậy để xác định được công việc cần làm thì đàu tiên chúng ta cần nói lại một số lệnh mà PBS cung câp cho người dùng để thao tác với Job.

Lênh đầu tiên chúng ta cân quan tâm chính là lệnh qsub, đây là lệnh cho phép người dùng đệ trình một công việc với hệ thống. Công việc được đệ trình phải được viết dưới dạng một file kịch bản từ trước. Tuy nhiên một khó khăn đối với người sử dụng lệnh này chính là việc có quá nhiều tham số cho lệnh và các tham số này đôi khi là rất xa lạ ngay cả với những người sử dụng chuyên nghiệp chứ đừng nói gì đên những người dùng không chuyên nghiệp.

Lênh tiếp theo mà chúng ta quan tâm là lệnh qalter, đây là lệnh cho phép người

dùng sử dụng để điều chỉnh các tham số về công việc mà mình đã đệ trình trước đây bằng lệnh qsub. Đây là một lệnh cũng rất quan trọng, lệnh này cho phép người dùng có một khả năng mềm dẻo hơn trong quá trình thực thi các công việc của mình.

Cũng như các thao tác khác chúng ta đôi khi cũng muốn xóa bỏ đi những công việc mà mình đã đệ trình trước đây(có thể vì một lý do nào đó) thì lệnh qdel sẽ đáp ứng nhu cầu này. Lệnh này cho phép người dùng xóa bỏ một công việc đã được đệ trình từ trước nhưng vẫn chưc được thực hiện.

Lệnh tiếp theo mà chúng ta quan tâm chính là lệnh qhold, đây là lệnh cho phép người dùng đưa một công việc nào đó vào trạng thái nằm chờ vô thời hạn cho đến khi công việc này được giải phóng bằng lệnh qrsl. Lệnh này cho phép chúng ta có thể bố chí chạy các job của mình một cách linh hoạt tùy thuộc vào các điều kiện ngoại cảnh tác động như có khi một công việc có thể được thực hiện ngay nhưng kết quả chúng ta lại chưa cần ngay thi chúng ta có thể sắp xếp cho công việc đó dừng lại để cho một công việc khác co kết quả quan trọng hơn(tại thời điểm này) được thực hiện trước.

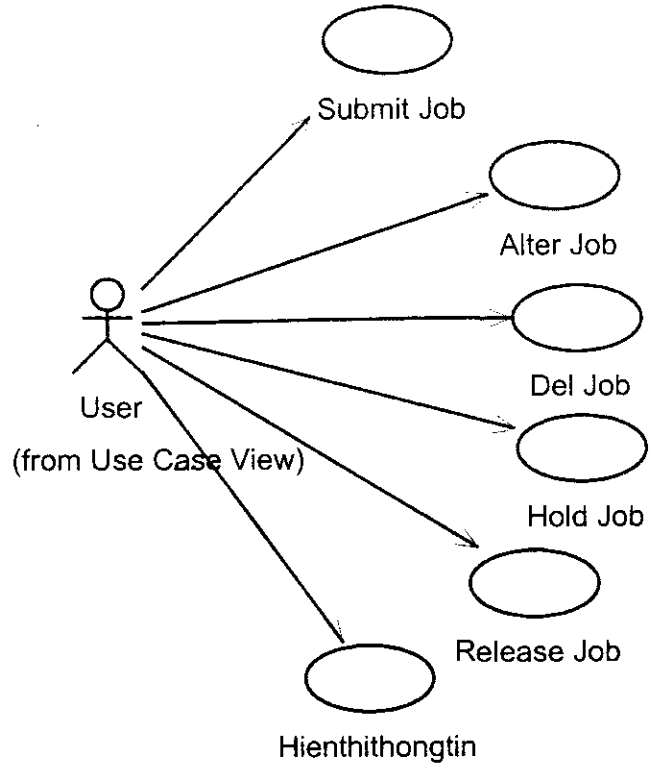
Ngược với lệnh qhold là lệnh qrsl, lệnh này dùng để giải phóng các công việc bị giữ bởi lệnh qhold.

Ngoài ra đôi khi người dùng muốn xem thông tin về các công việc của mình mà vẫn nằm trong hàng đợi chờ thực hiện, khi đó thì lệnh qstat sẽ đáp ứng nhu cầu này. Lệnh này cho phép chúng ta có khả năng xem được các thông tìn của các công việc đang nằm trong hàng đợi của hệ thống..

Như phân tích ở trên chúng ta đã thấy được một cách tổng quan các chức năng mà bộ công cụ User Tool cần cung cấp cho người sử dụng, tương ứng với mỗi lệnh ở trên là một chức năng mà bộ công cụ sẽ cung cấp cho người sử dụng nhưng ở một mức cao hơn, cho phép người dùng có thể thực hiện các lệnh này nhưng họ không hề biết đang thao tác với PBS trong môi trường làm việc song song mà chỉ như đang ngồi trên một máy tính đồng nhất và thực thi các lệnh để chạy công việc của mình. Như vậy bộ công cụ User Tool sẽ cần có các chức năng chính là submit job, alter job, hold job, release job, del job và hiển thị thông tin về job. Để thực hiện được các chức năng này trong bộ User Tool chúng ta sẽ sử dụng thư viện pbs_ifl do PBS cung cấp để giao tiếp với phần PBS bên dưới. pbs_ifl là thư viện do PBS cung cấp hỗ trợ cho người lập trình có thể giao tiếp với phần pbs_server ở bên dưới, thông qua thư viện này chúng ta có thể tiến hành viết lại các lệnh của PBS với các tham số đơn giản vầ dễ sử dụng hơn.

5.1.2 Phân tích thiết kế

Biểu đồ Use case của toàn bộ hệ thống:



Hình 5-1 Tổng thể các chức năng của User Tool

5.1.2.1 Chức năng Submit Job

Mô tả : Người sử dụng sử dụng chức năng này để đệ trình một công việc với hệ thống.

Luồng sự kiện thông lệ

Use case này được khởi tạo khi người dùng lựa chọn chức năng submit job. Sau khi người sử dụng chức năng này hệ thống sẽ yêu cầu người dùng đưa đường dẫn đến file kịch bản và hệ thống cũng yêu cầu người dùng nhập vào tên của file mã nguồn đã được dịch rồi. Ngoài hai thông tin bắt buộc trên thì hệ thong cho phép người dùng tự nhập các tham số liên quan đến công việc của mình hoặc lựa chọn các tham số mặc định sau đó hệ thống sẽ tự sinh ra file kịch bản và đệ trình công việc với hệ thống.

Use case kết thúc khi sau khi việc đệ trình công việc diễn ra xong, thông tin về Job sau đó sẽ được cập nhật vào cơ sở dữ liệu.

Luồng sự kiện biến thể

Luồng sự kiện biến thể là khi không tìm thấy file mã nguồn trong đường dẫn mà người submit job đưa ra, khi đó hệ thống sẽ hiển thị thông báo cho người sử dụng và yêu cầu người dùng nhập lại thông tin về đường dẫn đến file mã nguồn.

Ngoài ra nếu người dùng không nhập tên file kịch bản thì hệ thống cũng sẽ thông báo yêu cầu người dùng phải nhập tham số này vào.

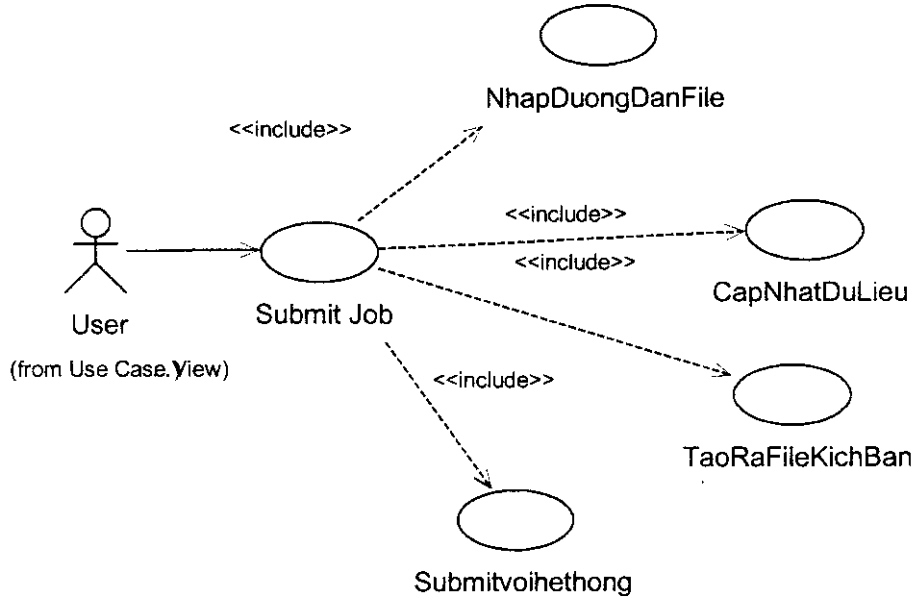
Điều kiện khởi đầu

Không có

Điều kiện kết thúc

Có thêm job do người dùng đệ trình được đưa vào hàng đợi

Use case diagram



Hình 5-2 Biểu đồ Use case của chức năng Submit Job

5.1.2.2 Chức năng Alter Job

Mô tả: Người dùng dùng chức năng này để điều chỉnh các thuộc tính về một công việc thuộc quyền sở hữu của mình

Luồng sự kiện thông lệ

Use case này được bắt đầu khi người dùng chọn một công và chọn chức năng Alter job.

Sau khi người dùng lựa chọn chức năng này hệ thống sẽ cho hiển thị các thông tin

về thuộc tính của công việc mà người sử dụng đã lựa chọn để người dùng có thể dễ dàng hơn trong khi điều chỉnh tham số mới cho công việc này.

Sau đó hệ thống sẽ yêu cầu nhập vào các giá trị mới của các thuộc tính của công việc mà chúng ta muốn thay đổi, sau khi nhập xong người dùng nhấn phím OK để xác nhận sự đồng ý.

Sau khi người dùng xác nhận đồng ý hệ thống sẽ cập nhật lại các thuộc tính này của công và kết thúc Use case này

Luồng sự kiện biến thể

Luồng sự kiện biến thể xuất hiện khi người dùng lựa chọn chức năng này nhưng không chỉ rõ công việc cụ thể nào khi đấy hệ thống sẽ yêu cầu người dùng phải lựa chọn một công việc nào đó.

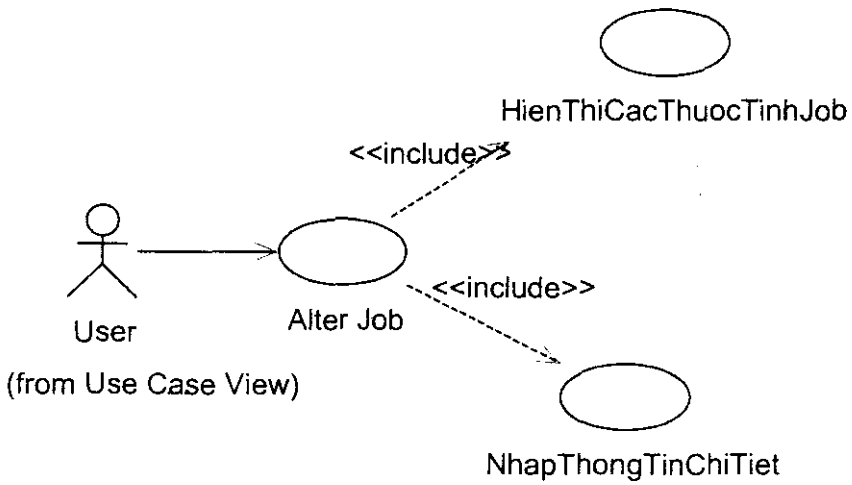
Điều kiện khởi đầu

Phải có công việc ở trong hàng đợi, và công việc dùng để điều chỉnh tham không ở trong trạng thái running.

Điều kiện kết thúc

Các thuộc tính của cập nhật và đệ trình lại với hệ thống.

Use case diagram



Hình 5-3 Use case cho chức năng Alter Job

5.1.2.3 Chức năng Hold Job

Mô tả: người sử dụng dùng chức năng này để đưa một job vào trạng thái chờ đợi cho đến khi trạng thái này được giải phóng thì job mới sẵn sàng để đưa vào trạng thái chạy

Luồng sự kiện thông lệ

Chức năng này được kích hoạt khi người dùng chọn một job và kích vào chức năng hold job.

Hệ thống sẽ kiểm tra xem job tương ứng có ở trong trạng thái running và có thuộc quyền sở hữu của người dùng đó hay không, nếu thỏa mãn hai điều kiện trên thì job tương ứng sẽ được đưa vào trạng thái chờ đợi.

Nếu mọi việc diễn ra bình thường thì sau khi đưa công việc vào trạng thái chờ đợi hệ thống sẽ cập nhật lại vào cơ sở dữ liệu trạng thái mới của công việc.

Luồng sự kiện biến thể

Luồng sự kiện biến thể xuất hiện khi người sử dụng lựa chọn một job không thuộc quyền sở hữu của mình hoặc một job đang ở trong trạng thái running khi này hệ thống sẽ hiển thị thông bao lỗi cho người dùng và Use case này kết thúc.

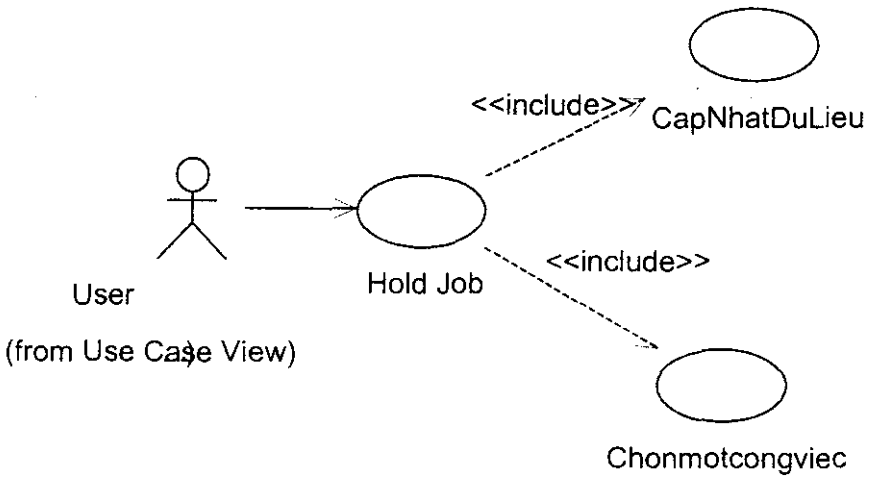
Điều kiện khởi đầu

Job được chọn phải tồn tại trong hệ thống, thuộc quyền sở hữu của người đưa ra yêu cầu và không ở trong trạng thái running

Điều kiện kết thúc

Hệ thống thông báo trở lại với người sử dụng kết quả thực hiện lệnh và job được đưa vào trạng chờ đợi.

Use case diagram:



Hình 5-4 Use case cho chức năng Hold Job

5.1.2.4 Chức năng Release Job

Mô tả: chức năng này được sử dụng để giải phóng các công ở trạng thái hold sang trạng thái sẵn sàng để được thực thi.

Luồng sự kiện thông lệ

Người sử dụng chọn một công đang ở trong trạng thái hold và nhấn vào chức năng Release job hệ thống sẽ giải phóng công tương ứng khỏi trạng thái hiện thời đồng thời cập nhật lại cơ sở dữ liệu.

Luồng sự kiện biến thể

Người sử dụng không có quyền giải phóng cho job này.

Người sử dụng yêu cầu giải phóng một công việc không ở trong trạng thái hold.

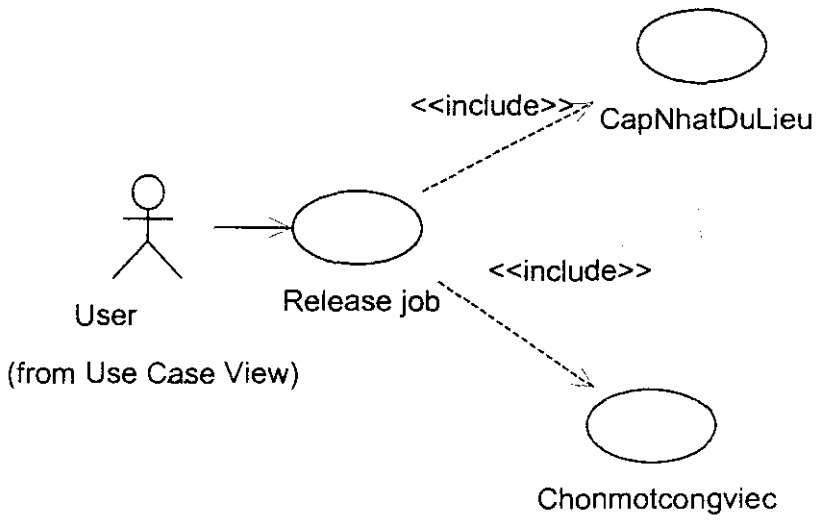
Điều kiện khởi đầu

Phải có job đang ở trạng thái hold với hold_type phù hợp với người sử dụng này.

Điều kiện kết thúc

Job tương ứng được giải phóng khỏi trạng thái hold và hệ thống thông báo lại kết quả cho người sử dụng.

Use case diagram



Hình 5-5 Use case cho chức năng Release Job

5.1.2.5 Chức năng Del Job

Mô tả: chức năng này dùng để xóa một công việc ra khỏi hàng đợi

Luồng sự kiện thông lệ

Use case này được khởi tạo khi người dùng lựa chọn một công việc sau đó kích vào chức năng Del job.

Hệ thống sẽ kiểm tra trạng thái của công việc nếu không có lỗi gì thì sẽ tiến hành xóa bỏ công việc đó.

Sau khi thực hiện hệ thống thông báo lại cho người dùng là đã xóa công việc tương ứng khỏi danh sách công việc trong hệ thống đồng thời cập nhật lại trong cơ sở dữ liệu.

Luồng sự kiện biến thể

Job được lựa chọn không thuộc quyền sở hữu của người dùng này, khi đó hệ thống sẽ thông báo lại và yêu cầu người dùng nhập lại thông tin về job hoặc hủy bỏ Use case này

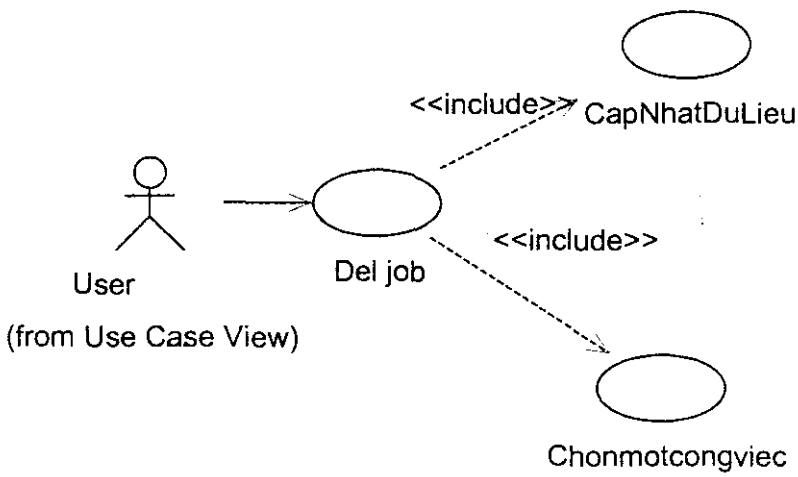
Điều kiện khởi đầu

Job được yêu cầu phải nằm trong một hàng đợi nào đó của hệ thống

Điều kiện kết thúc

Job được xóa khỏi hệ thống, nếu trước lúc xóa job ở trạng thái running thì một số tài nguyên của hệ thống sẽ được giải phóng

Use case diagram



Hình 5-6 Use case cho chức năng Del Job

5.1.2.6 Chức năng Show Info Detail

Mô tả use case

Use case này cho phép người dùng xem chi tiết các thông tin về công việc mà mình đã lựa chọn, chức năng này sẽ giúp cho người dùng quản lý tốt hơn các công việc của mình, có khả năng nắm được trạng thái và các tài nguyên mà công việc yêu

cầu.

Luồng sự kiện thông lệ

Use case được kích hoạt khi người sử dụng chọn một công việc trong danh sách các công việc của mình và lựa chọn chức năng hiển thị thông tin về công việc.

Hệ thống sẽ lấy các thông tin liên qua đến công việc mà người sử dụng yêu cầu và hiển thị thông tin cho người sử dụng.

Use case kết thúc khi người dùng xem xong và đóng chức năng này lại.

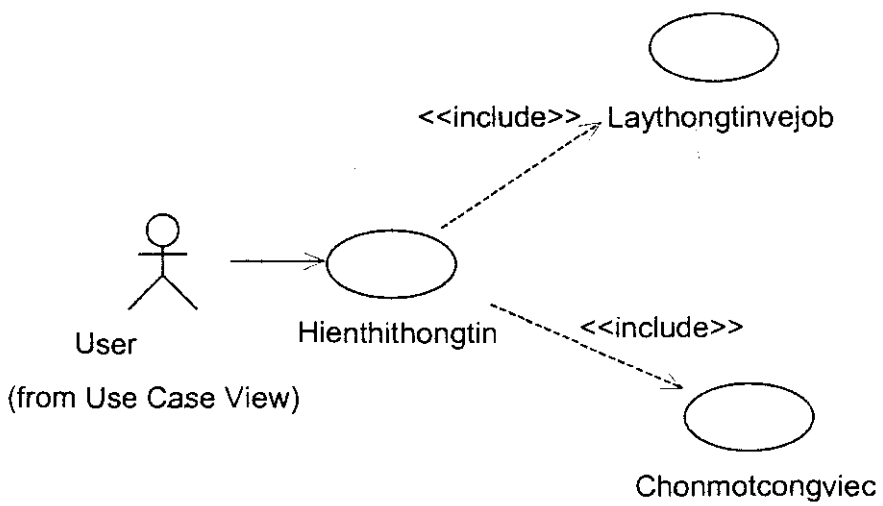
Luồng sự kiện biến thể

Luồng sự kiện biến thể xảy ra khi người dùng không lựa chọn một công việc cụ thể nào mà vẫn yêu cầu hiển thị thông tin về công việc, khi đó hệ thống sẽ hiện thông báo và yêu cầu người dùng lựa chọn một công việc trước khi lựa chọn chức năng này.

Điều kiện khởi đầu

- Phải có một công việc nào đó trong hàng đợi.
- Điều kiện kết thúc
- Không có

Use case diagram



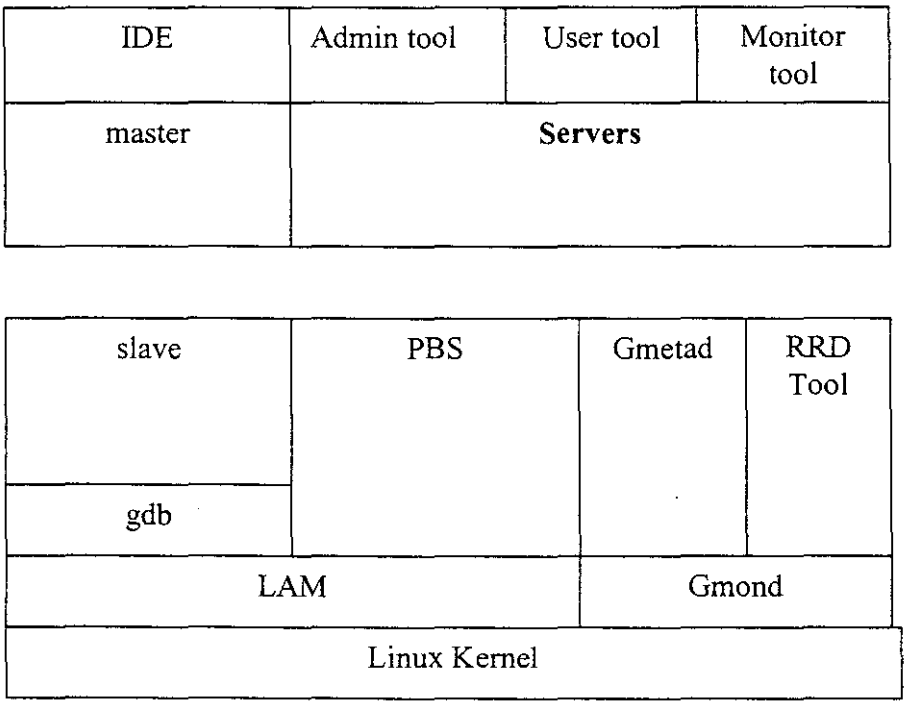
Hình 5-7 Use case cho chức năng Show Info Detail

5.2 Kiến trúc hệ thống và cài đặt các thành phần

5.2.1 Kiến trúc hệ thống

5.2.1.1 Kiến trúc của toàn bộ hệ thống BKluster

Như chúng ta đã biết hệ thông BKluster được xây dựng dựa trên gói phần mềm nguồn mở PBS. Nhờ việc sử dụng những tính năng phân tải rất ổn định của gói phần mềm này sẽ đảm bảo cho hệ thống chạy ổn định và có hiệu quả. Tuy nhiên do đặc trưng của PBS là phân tải và thực thi công việc nên nếu chúng ta chỉ sử dụng PBS thì sẽ rất khó trong việc theo dõi hệ thống vì vậy trong hệ thống BKluster có tích hợp thêm thành phần theo dõi thông tin hệ thống và thành phần này cũng dựa trên một gói phần mềm nguồn mở là Ganglia :



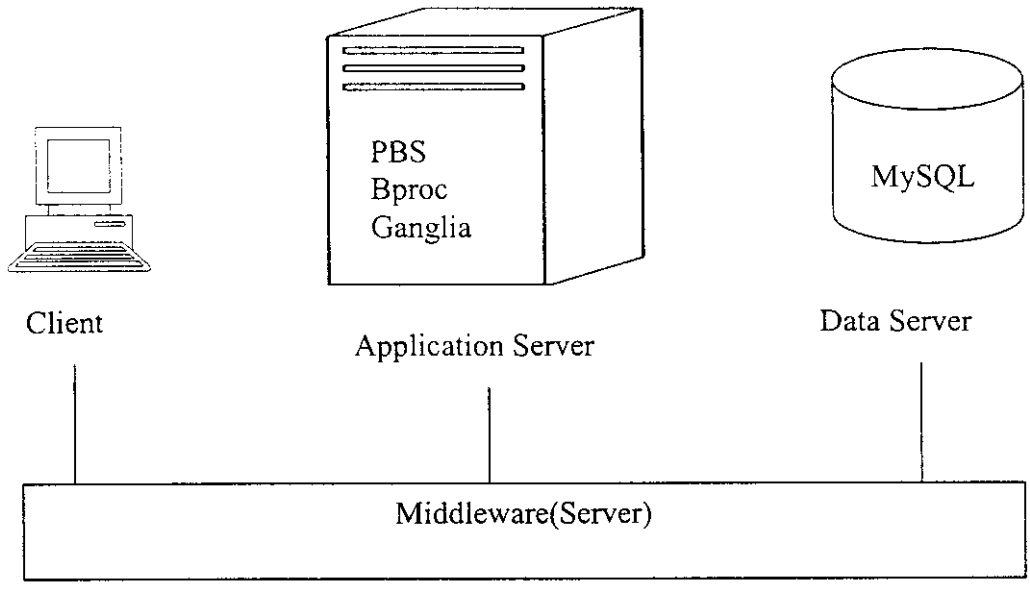
Hình 5-8 Vị trí các gói phần mềm mã nguồn mở trong Bkluster

Trong hình vẽ trên chúng ta thấy rằng hệ thống BKluster bao gồm 4 thành phần chính là:

- Thành phần cho những người phát triển phần mềm: đây là thành phần chứa bộ gdb, cho phép người dùng có thể sử dụng để soạn thảo, dich và debug lỗi cho một chương trình song song.

- Thành phần thứ hai là thành phần quản trị hệ thống: thành phần này gồm hai phần một phần server viết bằng C được đặt tại một máy làm Server, đây như là một thành phần midware giữa PBS và thành phần giao diện ở Client cung cấp cho người dùng. Thành phần này gồm hai chức năng chính là quản trị người dùng và cầu hình hệ thống. Việc quản trị người dùng chính là việc cụng cấp tài khoàn và mật khẩu cho người dùng, theo dõi việc đăng nhập hệ thống của người dùng. Còn việc cấu hình hệ thống, đây là nhiệm vụ chính của thành phần này, chính là việc thiết lập cấu hình vật lý cho hệ thống như là thiết lập hệ thống gồm bao nhiêu nút tính toán, tài nguyên của hệ thống ra sao... và việc cấu hình này phải đảm bảo sao cho hệ thống hoạt động ổn định.
- Thành phần thứ ba là thành phần dùng để thực thi các thao tác với các Job của người dùng. Chức năng chính của thành phần này là cung cấp cho người dùng sử dụng dich vụ của hệ thống BKluster một môi trường làm việc dễ dàng và thuận tiện trong quá trình thực thi các công việc song song của mình.
- Thành phần cuối cùng là thành phần dùng để thoi dõi các thông tin về hệt thống. Thành này cũng có một Server chạy ngầm trước và gửi các thông tin về hệ thống cho Client thông qua một cổng socket.

Trên đây chúng ta đã điểm qua toàn bộ các thành phần của hệ thống BKCluser, về mặt kiến trúc thì hệ thống BKluster được xây dựng theo mô hình Client-Server. Trong đó thành phần Client là chỉ đóng vai tro làm giao diện với người sử dụng và thu nhập thông tin người dùng. Các thông tin mà người dùng yêu cầu hệ thống thực hiện sau đó sẽ được truyền cho thành phần midware (chính là thành phần Server mà chúng ta xây dựng), tại đây thông tin bắt đầu được sử lý và đưa vào thực thi tại các thành phần ở bên dưới của hệ thống (PBS, gmeta...) .Sau khi thực hiện xong thì kết quả lại được trả lại Client cho người sử dụng có thể xem. Kiến trúc theo mô hình Client-Server của hệ thống



Hình 5-9 Mô hình Client-Server của hệ thống BKluster

Trên đây là kiến trúc tổng thể của toàn bộ hệ thống BKluster, trong phần sau chúng ta sẽ cùng đi vào phân tích chi tiết kiến trúc của bộ User Tool. Chúng ta sẽ xem xét với mô hình tổng thể như trên khi được áp dụng vào bộ User Tool sẽ gồm những thành phần như thế nào và hoạt động của từng thành phần cụ thể bên trong hệ thống.

5.2.1.2 Kiến trúc của bộ công cụ User Tool

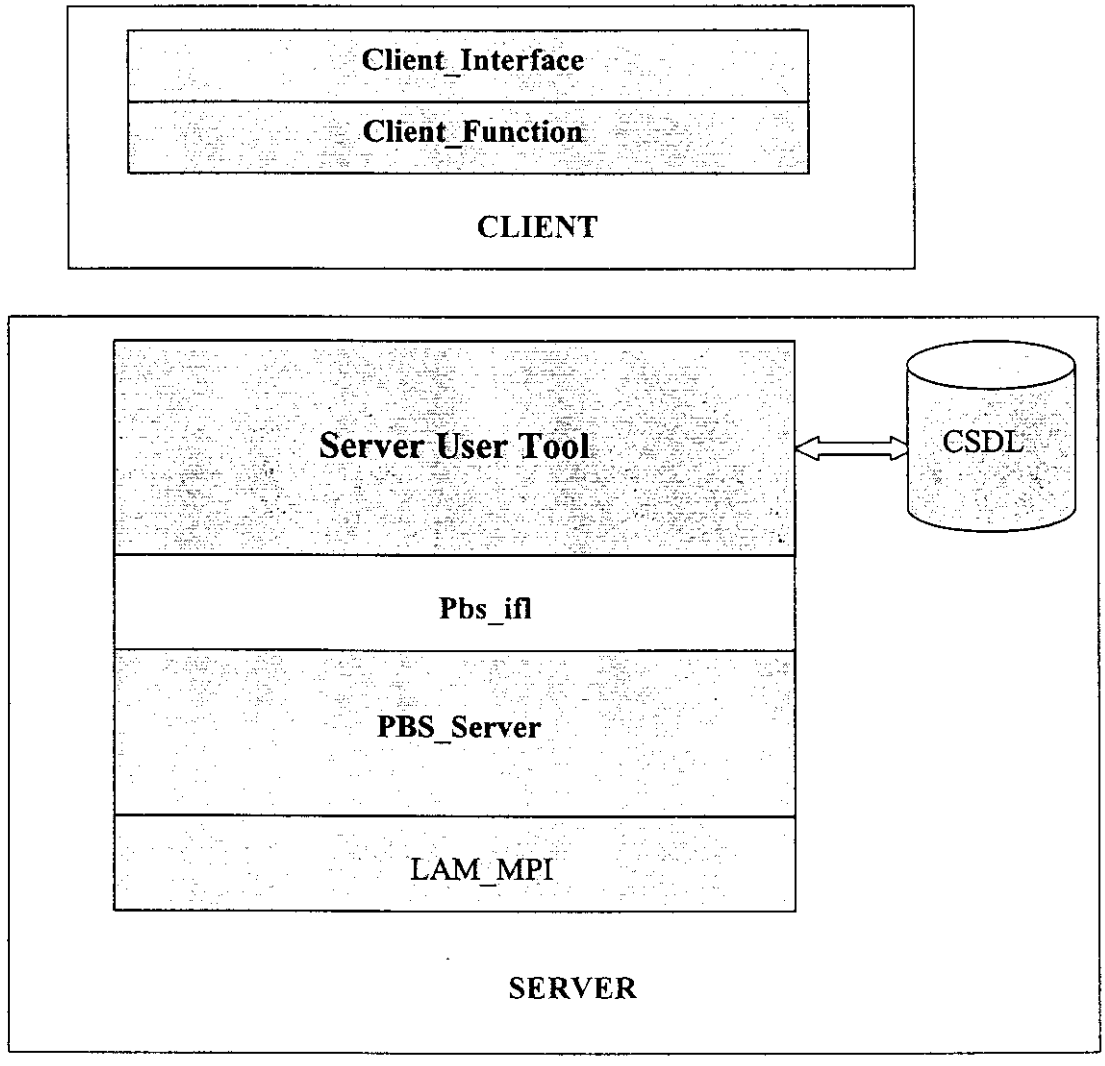
Bộ User Tool được xây dựng dựa với mục đích cung cấp cho người dùng một môi trường làm việc dễ dàng hơn với hệ thống BKluster. Bộ công cụ này được xây dựng vơi thành phần trung tâm là gói phần mềm PBS, mọi thao tác mà bộ cung cấp cho người dùng đều nhằm mục đích để người dùng giao tiếp một cách dễ dàng hơn vói thành phần PBS ở bên dưới.

Như trong mô hình ban đầu đã đề cập bộ User Tool là một phần troang hệ thống BKluster nên nó cũng phải có tuân theo mô hình của hệ thống BKluster. Nghĩa là bộ công cụ User Tool cũng được xây dựng theo mô hình Client-Server, trong đó máy Server sẽ có nhiệm vụ giao tiếp với phần PBS và thực hiên giao tiếp với Client để lấy về các yêu cầu của người dùng.

Phía Server để thực hiện việc giao tiếp với PBS thì chúng ta sẽ sư dụng bộ thư việc ifl do PBS cung cấp để thực thi các lệnh của người dùng còn giao tiếp với Client thông qua Socket. Việc phân chia thành mô hình Client-Server như thế này

sẽ đảm bảo tính độc lập của hệ thống với hệ điều hành. Bởi vì chỉ có Server là phải bắt buộc chạy trên hệ điều hành Linux, vì cần giao tiếp trực tiếp với PBS và đồng thời cũng yêu cầu phải cài PBS, còn phía Client chúng ta có thể chạy trên hệ điều hành Linux hay windows vì cả hai hệ điều hành này đều hỗ trợ cho việc giao tiếp qua socket.

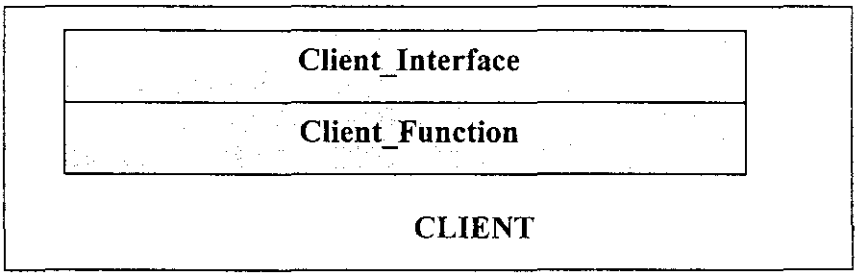
Ngoài ra việc phân tích thành hai phần, Client và Server như trên cũng đảm bảo cho tính độc lập sau này của hệ thống. Như vậy sau này khi chúng ta tiến hành nâng cấp một trong hai phần thì không cần thiết phải viết lại toàn bộ hệ thống. Thêm nữa việc sử dụng mô hình Client-Server sẽ làm cho kiến trúc hệ thống được rõ rang thuận lợi cho quá trình xây dựng hệ thống. Sau đây là mô hình kiến trúc của bộ User Tool :



Hình 5-10 Kiến trúc của User Tool

Như trong hình vẽ chúng ta cũng thấy hệ thống được chia làm hai phần là phần Client và Server như chúng ta đã phân tích ở trên, việc giao tiếp giữa hai phần này được thực hiện thông qua Socket. Tiếp theo chúng ta sẽ đi phân tích cụ thể nhiệm vụ và kiến trúc của từng phần.

Thành Phần Client



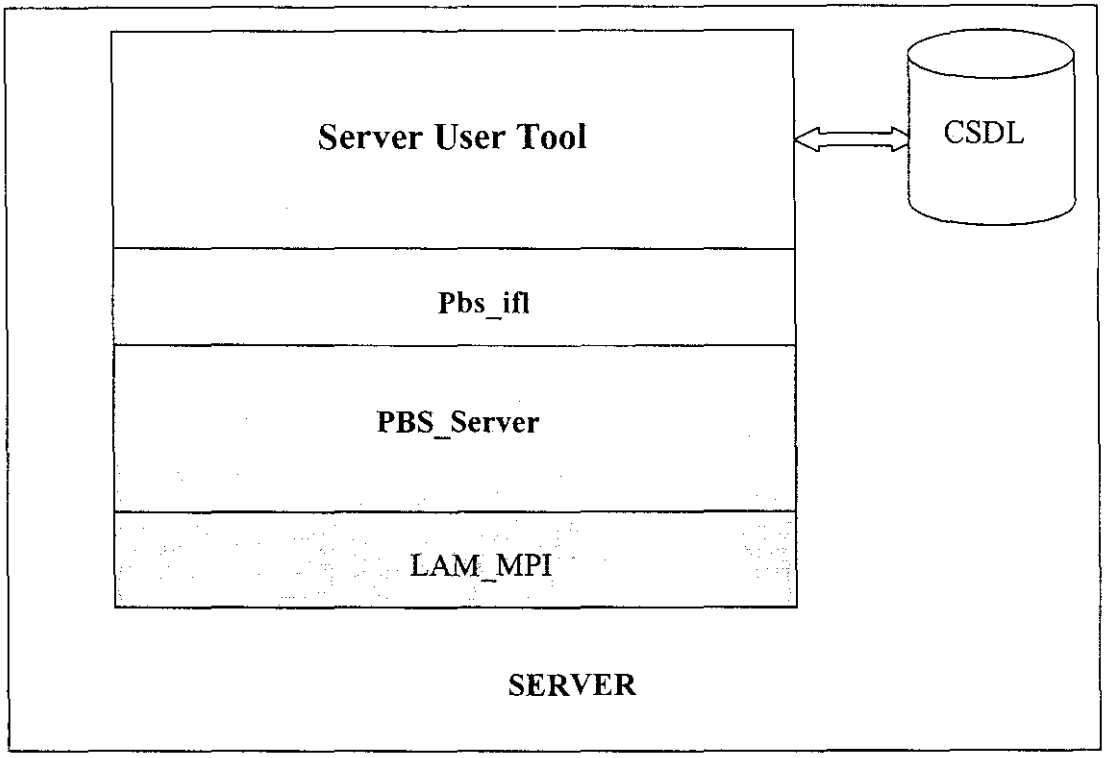
Hình 5-11 Thành phần Client

Đây là phần cài đặt trên các máy của người dùng. Nhiệm vụ chính của phần này là lấy về các tham số mà người dùng muốn đưa vào trong Job của mình sau đó truyền về cho Server.

Phần này được chia làm hai phần là Client_Interface và Client_Function. Phần Client_Interface có nhiệm vụ làm giao diện của toàn bộ hệ thống với người sử dụng, đây chính là nơi người sử dụng nhập các thông tin vào đồng thời nhận kết quả trả về. Sau khi người dùng nhập thông tin thi các thông tin này được truyền cho thành phần Client_Function ở bên dưới. Phần Client_Function là thành phần ở phía Client dùng để xử lý thô thông tin mà người dùng nhập vào trước khi truyền cho phía Server, đồng thời phần này cũng sẽ tiến hành xử lý thông tin được truyền về từ Server trước khi hiển thị cho người dùng xem trên màn hình Client. Ngoài ra Client_Function còn đảm nhận việc truyền thông với phần Server. Việc chúng ta để cho thành phần Client thực hiện sử lý một số thông tin thô ban đầu sẽ phát huy được sức mạnh tính toán của các máy Client đồng thời cũng giảm tải cho Server.

Phần Client này được viết bằng ngôn ngữ QT và hoàn toàn có thể chạy độc lập với nền hệ điều hành.

ThànhPhần Server



Hình 5-12 Thành phần Server

Toàn bộ phần Server này được chạy trên nền hệ điều hành Linux, trên một hệ thống tính toán lưới đã được cài đặt bộ lập lịch PBS.

Chúng ta có thể thấy rõ phần này gồm có hai phần chính là phần Server User Tool và PBS_Server.

PBS_Server đây chính là PBS đã được cài đặt lên hệ thống tính toán phân cụm của chúng ta và được dùng để tiến hành lập lịch cho các công việc song song sau này được đệ trình với hệ thống. Còn thành phần LAM_MPI bên dưới là môi trường truyền thông điệp giữa các tiến trình song song của một công việc, đây là một tiến trình ngầm của hệ thống.

Thành phần Server User Tool là thành phần do chúng ta xây dựng dùng để thực thi các yêu cầu từ phía người dùng để giao tiếp với PBS, thành phần này sẽ giao tiếp với PBS_Server thông qua thư viện pbs_ifl do PBS cung cấp để thực hiện các thao tác mà người dùng yêu cầu. Ngoài ra thành phần này còn có nhiệm vụ truy suất vào trong cơ sở dữ liệu để ghi lại những công việc mà một người nào đõ đã thực hiện, cập nhật trạng thái công việc.

5.3 Thực hiện cài đặt các thành phần của hệ thống

Từ kết quả phân tích thiết kế cùng với kiến trúc các thành phần của hệ thống đã được phân tích ở trên, trong phần này sẽ đưa ra chức năng cụ thể của từng thành phần trong mộ hình Client-Server trong mõi chức năng mà hệ thông cung cấp cho người sử dụng.

Để cài đặt bộ User Tool chúng ta sẽ sử dụng thư viện pbs_ifl do PBS cung cấp để tương tác với hệ thống phần PBS ở bên dưới và thư viện mysqlclient.h dùng để tương tác với cơ sơ dữ liệu. Ngôn ngữ lập trình được sử dụng là C (dùng để xây dựng Server) và QT (dùng để xây dựng Client).

5.3.1 Chức năng đệ trình công việc

Đây là chức năng Submit Job đã được nhắc đến ở trên, để tiện cho việc theo doi chúng ta sẽ nhắc lại kịch bản thông lệ của chức năng này.

Chức năng này được kích hoạt khi người dùng có nhu cầu đệ trình một công việc song song với hệ thống. Khi người dùng đệ trình một công việc với hệ thống thì hệ thống sẽ cho phép người dùng lựa chon các tham số của công việc này để chạy. Yêu cầu là công việc được đệ trình phải có một file mã nguồn đã được dich rồi để chạy.

Từ việc phân tích Use case ở trên chúng ta sẽ phân tich từng phần công việc cho phía Client và Server.

Như trong kịch bản của chức năng đã nói chùng có thể nhận ra trong chức năng này thì phía Client là nơi nhân dữ liệu về các tham số các thuộc tính của công việc và sau khi có xử lý một số thông tin sơ bộ sẽ truyền về cho phía Server để đệ trình công việc với phần PBS_Server.

Phía Server sau khi nhận được yêu cầu đệ trình công việc từ phía Client sẽ chờ đọc về các thuộc tính về công việc do Client truyền về sau đó sẽ tự động sinh ra file kịch bản cho công việc rồi tiến hành đệ trình công việc này vơi PBS_Server. Nếu việc đệ trình thành công thì các thông tin mới về công việc này sẽ được cập nhật vào trong cơ sở dữ liệu, được đặt ở Server để tiện cho việc quản lý công việc sau này. Còn nếu việc đệ trình gặp lỗi thì sẽ hiển thị thông báo lên trên cho người sử dụng.

Phía Server sẽ đệ trình công việc với PBS_Server thông qua hàm pbs_submit của thư viện pbs_ifl.

5.3.2 Chức năng điều chỉnh tham số của công việc

Đây là chức năng tương ứng với chức nưng Alter Job đã được phân tích ở trên, chức năng này cho phép người sử dụng có thể điều chỉnh các tham số về công việc mà mình đã đệ trình với hệ thống trước đó. Để tiện cho việc phân tích chức năng này theo mô hình Client-Server chúng ta sẽ nhắc lại một cách sơ qua về kịch bản của chức năng này.

Chức năng này được kích hoạt khi người dùng lựa chọn từ danh sách các công việc của minh trong hàng đợi và chọn chức năng điều chỉnh các tham số của công việc. Sau khi người dùng lựa chọn chức năng này hệ thống sẽ hiển thị các thông tin liên quan đến công việc này và yêu cầu người dùng nhập vào các tham số mới cho công việc này. Sau khi người dùng nhập các thông tin về công việc thì xác nhân lại cho hệ thống những thông tin mơi về công việc để hệ thống cập nhật lại với PBS.

Luông sự kiện ngoại lệ xảy ra khi người dùng sau khi xem xong thông tin về công việc này thi không tiến hành điều chỉnh tham số công việc khi đấy hệ thống sẽ hủy bỏ phiên làm việc này và không tiến hành điều chỉnh tham số của công việc vừa được yêu cầu nữa. Từ việc phân tích Use case ở trên chúng ta sẽ cùng xem xét phần việc của Client và Server trong chức năng này của hệ thống BKluster.

Client: cũng giống như trong chức năng đệ trình công việc, ở chức năng này thì Client là nơi giao tiếp giữa người sử dụng và hệ thống vì vậy Client có nhiệm vụ thu nhân thông tin từ phía người sử dụng và hiển thị những thông tin từ hệ thống cho người sử dụng biết. Như trong kịch bản của chức năng này thì chúng ta có thể thấy rằng Client làm nhiệm vụ nhân thông tin về các thuộc tính mới mà người sử dụng yêu câu cho công việc của mình và sau đó đưa về cho phía Server. Ngoài ra trước khi nhận các thông tin này thì Client con phải truyền về cho Server định danh của công việc mà người dùng muốn điều chỉnh tham số.

Thêm nữa Client ở trong chức năng này còn có nhiệm vụ hiển thị các thông tin về cồng việc mà người dùng muốn điều chỉnh tham số để tiện cho người sử dụng trong quá trình thay đổi các tham số của công việc, các thông tin của công việc ở đây thực chất chính là nội dung file kịch bản của công việc để thông qua đó người dùng có thể nắm được những thông tin về công việc mà mình đã đệ trình trước đây.

Server: trong chức năng này Server sau khi nhận được yêu cầu điều chỉnh tham số của công việc cùng với định danh của công việc sẽ tiến hành tìm kiếm file kịch

bản của công việc và sau đó truyên nội dung của file này cho phía Client để Client hiển thị thông tin này lên cho người sử dụng biết.

Sau khi truyền nội dung của file kịch bản cho người sử dụng thì Server sẽ "đợi" kết quả trả về từ phía Client, kết quả ở đây chính là các thông tin mới về thuộc tính của công việc. Sau khi nhận được các tham số mới này từ phía Client thì Server sẽ gọi hàm pbs_alterjob để điều chỉnh lại các tham số của công việc này với PBS. Hàm pbs_alterjob là một hàm do thư viện pbs_ifl của PBS cung cấp nhằm cho phép người lập trình có thể tiến hành điều chỉnh các tham số của một công việc ngay trong chương trình.

5.3.3 Chức năng Tạm dừng công việc-Hold Job

Đây là chức năng cho phép người dùng thay đổi trạng thái của công việc mà mình đã đệ trình trước đây, cụ thể ở đây là người dùng sẽ sử dụng chức năng này để chuyển trạng thái của một công việc về trạng thái chờ đợi vô thời hạn, nghĩa là công việc đây sẽ không thể được thực hiện, cho đên khi người dùng giải phóng công việc khỏi trạng thái này.

Chức năng này được kích hoạt khi người dùng lựa chọn một công việc va yêu cầu hệ thống đưa công việc này vào trạng thái "hold". Hệ thống sau khi nhận được yêu cầu từ phía người dùng sẽ kiểm tra lại xem yêu cầu này có hợp lệ hay không. Vì công việc có thể đưa vào trạng thái hold phải là công việc đang trong trạng thái chờ đợi và không ơ trong trạng thái hold rồi. Nếu sau khi kiểm tra không có vấn đề gì xảy ra thì hệ thống sẽ đưa công việc này vào trạng thái hold sau đó sẽ cập nhật vào trong cơ sở dữ liệu về trạng thái mới của công việc.

Trên đây là tóm tắt lại kịch bản của Use case này, trên cơ sở kịch bản này chúng ta sẽ phân tích công việc của Client và Server trong chức năng này.

Cũng giống như trên thì Client ở đây sẽ sử dụng để thu thập thông tin tử phía người sử dụng. Client có nhiệm vụ đưa về cho Server những thông tin về công việc mà người dùng muốn đưa vào trạng thái hold. Ngoài ra Client còn làm thêm nhiệm vụ là kiểm tra tính hợp lệ của yêu cầu mà người sử dụng đưa ra, nghĩa là việc kiểm tra trạng thái của công việc được thực hiện hoàn toàn ở bên Client. Sở dĩ chúng ta thực hiện việc kiểm tra này ở phía Client thay vì truyền về Server rồi mới kiểm tra là vì ngay ở Client chúng ta cũng đã biết được trạng thái của công việc, do trạng thái này đã được Server truyền cho Client trước đó. Và việc thực hiện như thế này sẽ giảm tải cho Server.

Trong chức năng này Server sau khi nhân được định danh của công việc mà người dùng muốn đưa vào trạng thái nằm chờ(hold) sẽ gọi hàm pbs_holdjob để đưa công việc vào trạng thái chờ đợi, sau đó sẽ tiến hành cập nhật trạng thái của công việc này vào cơ sở dữ liệu. Như vậy chúng ta thấy công việc của phía Server trong chức năng này có vẻ tương đối đơn giản, sở dĩ có được điều này là do phía Client chúng ta đã xử lý sơ bộ thông tin mà người dùng nhập vao và như vậy khi Server nhận được thông tin về công việc thì mặc nhiên công việc này đã thỏa mãn mọi điều kiện để có thể xử lý nên không phải làm thêm gì để kiểm tra trạng thái của công việc nữa.

Trong chức năng này chúng ta có sử dụng hàm pbs_holdjob do PBS cung cấp.

5.3.4 Chức năng thực thi lại công việc-Release job

Đây là chức năng ngược với chức năng hold job ở trên, chức năng này cho phép người dùng giải phong các công việc khỏi trạng thái hold và đưa trở lại trạng thái sẵn sàng thực hiện khi có đủ điều kiện về tài nguyên.

Chức năng này được kích hoạt khi người dùng lựa chọn một công việc của mình đang ở trong trạng thái hold và yêu cầu giải phóng công việc khỏi trạng thái này. Hệ thống sau khi kiểm tra tính hợp lệ của yêu cầu người dùng sẽ tiến hành thực hiện yêu cầu của người dùng, kết quả trả về là công việc này được đưa trở lại trạng thái sẵn sàng thực hiện và kết quả được cập nhật vào trong cơ sở dữ liệu.

Trên đây là toàn bộ kịch bản của hệ thống, bây giờ chúng ta sẽ phân chia công việc cho từng phần cụ thể

Tương tự như trong chức năng Hold Job ở trên, trong chức năng này Client có nhiệm vụ lấy thông về công việc mà người dùng muốn giải phóng khỏi trạng thái hold và đưa trở lại hàng đợi với trạng thái sẵn sàng thực hiện trở lại, bên cạnh đó Client còn có nhiệm vụ kiểm tra cả tính hợp lệ của yêu cầu người dùng. Kiểm tra tính hợp lệ ở đây chính là chúng ta tiến hành kiểm tra xem công việc mà người dùng yêu cầu có ở trong trạng thái hold sẵn không, nếu yêu cầu người dùng không hợp lệ thì hệ thống sẽ thông báo cho người dùng và hủy bỏ phiên làm việc. Ngược lại nếu yêu cầu từ phía người dùng là hoàn toàn.

Phía Server trong chức năng này cũng tương tự như trong chức năng Hold Job ở trên, nghĩa là Server ở đây chỉ có nhiệm vụ đợi Client truyền về thông tin của công việc(cụ thể ở đây chính là định danh của công việc) sau đó sử dụng hàm pbs_rlsjob do PBS cung cấp để giải phóng trạng thái của công việc khỏi trạng thái

hold. Sau khi giải phóng trạng thái của công việc xong thì Server sẽ cập nhật lại trạng thái của công việc trong cơ sở dữ liệu.

Trong chức năng này phía Server có sử dụng hàm pbs_rlsjob của PBS cung cấp.

5.3.5 Chức năng xóa bỏ một công việc

Chức năng này nhằm giúp người sử dụng có khả năng xóa bỏ một công việc mà mình đã đệ trình trước đây nhằm tiết kiệm tài nguyên của hệ thống.

Về mặt phân chia công việc cho Client và Server thì ở chức năng này cũng tương tự như trong hai chức năng trên. Nghĩa là Client cũng có nhiệm vụ nhận về yêu cầu của người dùng và kiểm tra tính hợp lệ của yêu cầu này. Còn Server khi nhận được định danh của công việc cần xóa bỏ sẽ mặc nhiên công nhận là công việc này đã thỏa mãn mọi yêu cầu để có thể xóa bỏ và sử dụng hàm pbs_deljob để xõa bỏ công việc khỏi hàng đợi, sau đó tiếp tục xỏa bỏ công việc trong cơ sở dữ liệu.

5.3.6 Chức năng hiển thị thông tin chi tiết của công việc - Show Info Detail

Đây là chức năng cho phép người dùng có thể xem thông tin chi tiết về các công việc mà mình đã đệ trình và công việc này vẫn chưa được thực hiện.

Chức năng này được kích hoạt khi người dùng lựa chọn từ danh sách công việc của mình trong hang đợi một công việc và yêu cầu xem thông tin chi tiết về công việc này. Sau khi nhận được yêu cầu từ phía người dùng hệ thống sẽ tiến hành lấy thông tin về công việc này và hiển thị laic cho người dung xem. Các thông tin trả về cho người dùng sẽ là những thông tin về các thuộc tình của công việc, các thông tin liên quan đên ngày giờ đệ trình, ngày giờ công việc được chạy....

Trên đây chúng ta đã nói về công việc của toàn hệ thống trong chức năng này, tiếp theo chúng ta sẽ cùng xem xét đến từng phần công việc cụ thể cho Client và Server.

Client: cũng giống như các chức năng trước của hệ thống, client có nhiệm vụ đầu tiên là làm nơi cho phép người dùng giao tiếp với hệ thống. Cụ thể trong chức năng này Client có nhiệm vụ lấy thông tin định danh công việc mà người dùng muốn xem thông tin chi tiết và sau khi truyền thông tin này về cho Server thì nằm đợi kết quả trả về từ Server để hiển thị thông tin nay cho người sử dụng.

Server: Trong chức năng này Server sau khi nhận được thông tin của Client về định danh của công việc và có yêu cầu từ Client cân xem thông tin chi tiết về công

việc này thì sẽ kết nối đến PBS_Server để lấy thông tin về công việc sau đó truyền về cho Client để hiển thị lên cho người dùng. Để lầy thông tin về công việc Server dùng hàm pbs_statjob do PBS cung cấp để lấy thông tin về công việc mà người dùng yêu cầu.

Trong quá trình phân tích các chức năng ở trên chúng ta đã rất nhiều lần nhắc đến danh sách công việc của người dùng, và người dùng sẽ lựa chọn một công việc trong dánh sách này để thực hiện các thao tác của mình. Vậy danh sách đấy ở đâu ra và nó có được người sử dụng biết đến như thế nào? Thực tế là khi người dùng sử dụng bộ công cụ User Tool thì đầu tiên người dùng sẽ thấy một form giao diện chính của bộ công cụ này, trên form giao diện chính này sẽ có danh sách các công việc của người dùng đã được thực hiện trước đây và ngay sau khi người sử dụng đệ trình một công việc mới thì hệ thống sẽ cập nhật lại danh sách này.

Sau đây chúng ta sẽ phân tích cụ thể các phân công việc cho việc hiển thị danh sách công việc này cho người sư dụng.

Đâu tiên chúng ta cùng mô tả chung cho Use case này. Use case này được tự động kích hoạt ngay khi người sử dụng sử dụng các chức năng của bộ công cụ dành cho người dùng(User Tool) nó giúp cho người dùng có thể dễ dàng hơn trong quá trình kiểm soát các công việc mà mình đã thực hiện.

Trong Use case này Client có nhiệm vụ hiển thị thông tin lên cho người sử dụng, không cần thiết phải thu nhận bất ký dữ liệu nào được nhập vào từ người dùng. Tuy nhiên vì người sử dụng chỉ được phép theo dõi các công việc của mình nên trong Use case này Client sẽ phải truyền về cho Server thông tin về người dung, cụ thể ở đây chính là tên của người dùng trong hệ thống. Sau khi truyền tên người dùng cho Server thì Client sẽ đợi lấy kết quả trả về từ Server về các thông tin liên quan đến công việc mà người dùng đã thực hiện được và hiển thị lại cho người sử dụng. Ngoài ra sau một khoảng thời gian nhất định Client cũng phải gửi các yêu cầu cập nhật lại danh sách các công việc mà người dùng đã thực hiện, bởi vì có những công việc mới được người dùng thực hiện ngay trong phiên làm việc này. Thêm vào đó để hỗ trợ cho người dùng thì hệ thống cho phép người dùng tự động yêu cầu cập nhật lại danh sách nay mà không cần đợi đến thời gian định kỳ.

Server trong chức năng này có nhiệm vụ lắng nghe thông tin về người sử dụng sau đó lấy thông tin này để truy suất vào cơ sở dữ liệu lấy ra những công việc liên quan đến người dùng này và trả kết quả lại cho Client để hiển thị lại cho người sử dụng.

Trong hầu hết các chức năng của bộ công cụ dành cho người dùng chúng ta đều thấy nhắc đến các thao tác với cơ sở dữ liệu nhưng chúng ta chưa hề nhắc đến tổ chức cơ sở dữ liệu trong hệ thống của chúng ta. Trong hệ thống BKClusware nói chung và bộ công cụ dành cho người dùng nói riêng chúng ta dùng hệ quản trị cơ sở MySQL để quản trị cơ sở dữ liệu.

Cơ sở dữ liệu của hệ thống gồm ba bảng là

- Bảng user : đây là bảng dùng để quản trị người dùng, bảng này dùng để chứa các thông tin về người dùng. Các thông tin trong bang này bao gồm: uname, uid, pass, roleid(trường này dùng để quy định quyền của người dùng trong hệ thống)...
- Bảng Job : đây là bảng chứa các thông tin liên quan đến các công việc trong hệ thống. Bang này bào gồm các trường:
    - Jobid : đinh danh của công việc trong hệ thống.
    - uid : định danh của người dùng sở hữu công việc này
    - Script : tên file kịch bản của công việc này
    - State : trạng thái của công việc.
    - DateTime : ngày giờ công việc được thực hiện.
- role : đây là bảng chứa danh sách các quyền của người dùng đối với hệ thống. Bảng này chứa thông tin về định danh của quyền dành cho người dùng và mô tả thêm chi tiết về quyền tương ứng với định danh này.

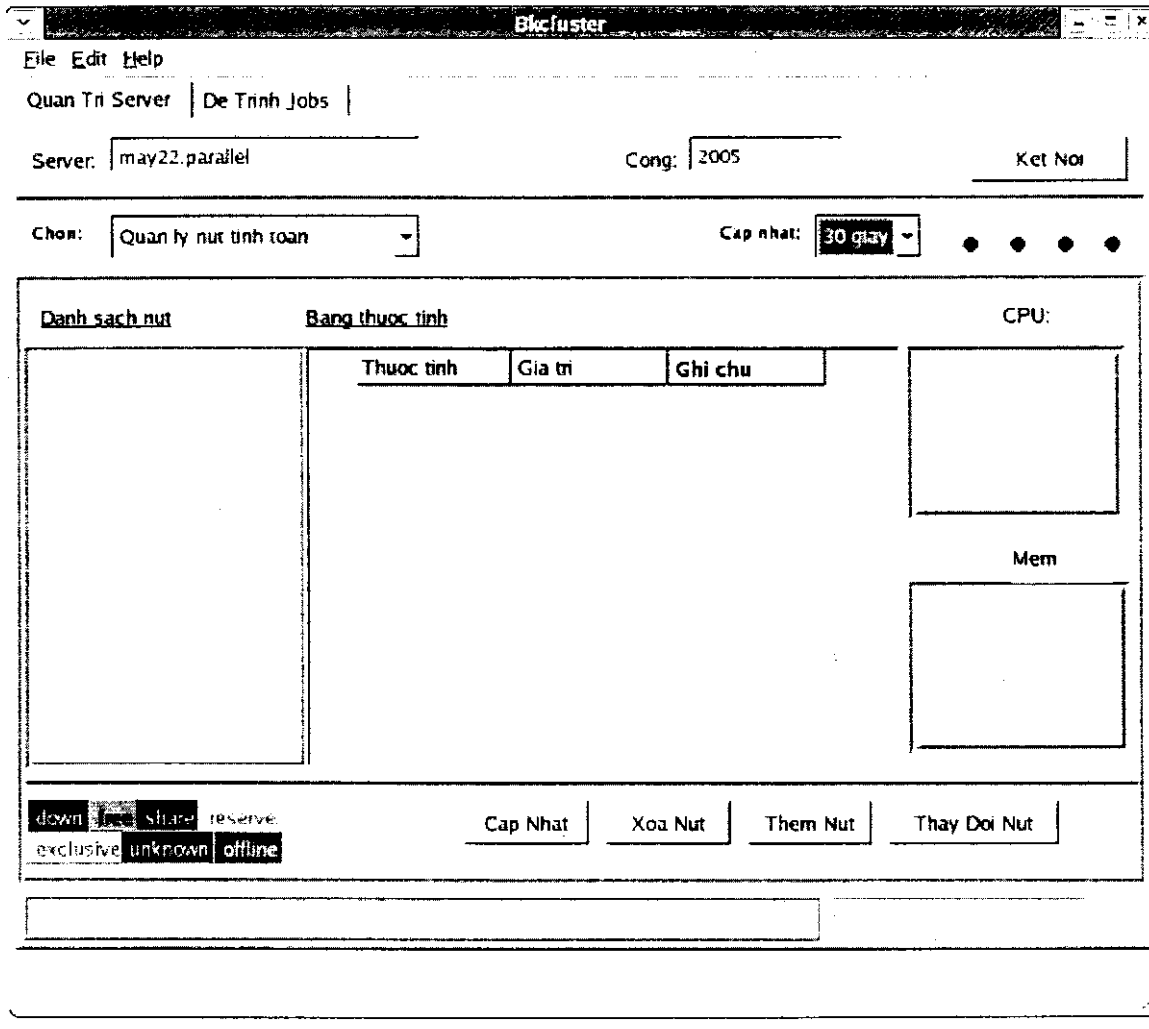
5.4 Kết quả đạt được và định hướng phát triển

5.4.1 Đánh giá kết quả đã đạt được

Sau quá trình thực tập trên trung tâm, cùng vơi các bạn trong nhóm HPC chúng em đã tích hợp thành công thêm vào hệ thống BKluster mà trung tâm đã xây dụng từ trước các tính năng cho phép các người dùng có thể giao tiếp một cách thuận tiện hơn với thành phần PBS ở bên dưới. Các tính năng được tích hợp thêm này cho phép người quản trị có thể ngồi từ xa để cấu hình toàn bộ một hệ thống tính toán phân cụm và tiến hành việc quản trị toàn bộ hệ thống, bên cạnh đó hệ thống BKluster cũng cho phép người dùng xem xét các thông tổng thể cũng như chi tiết về các hoạt động của toàn bộ hệ thống. Thêm nựa hệ thống BKluster cho phép người dùng có thể tiến hành thao tác với các bài toán yêu câu đến việc tính toán

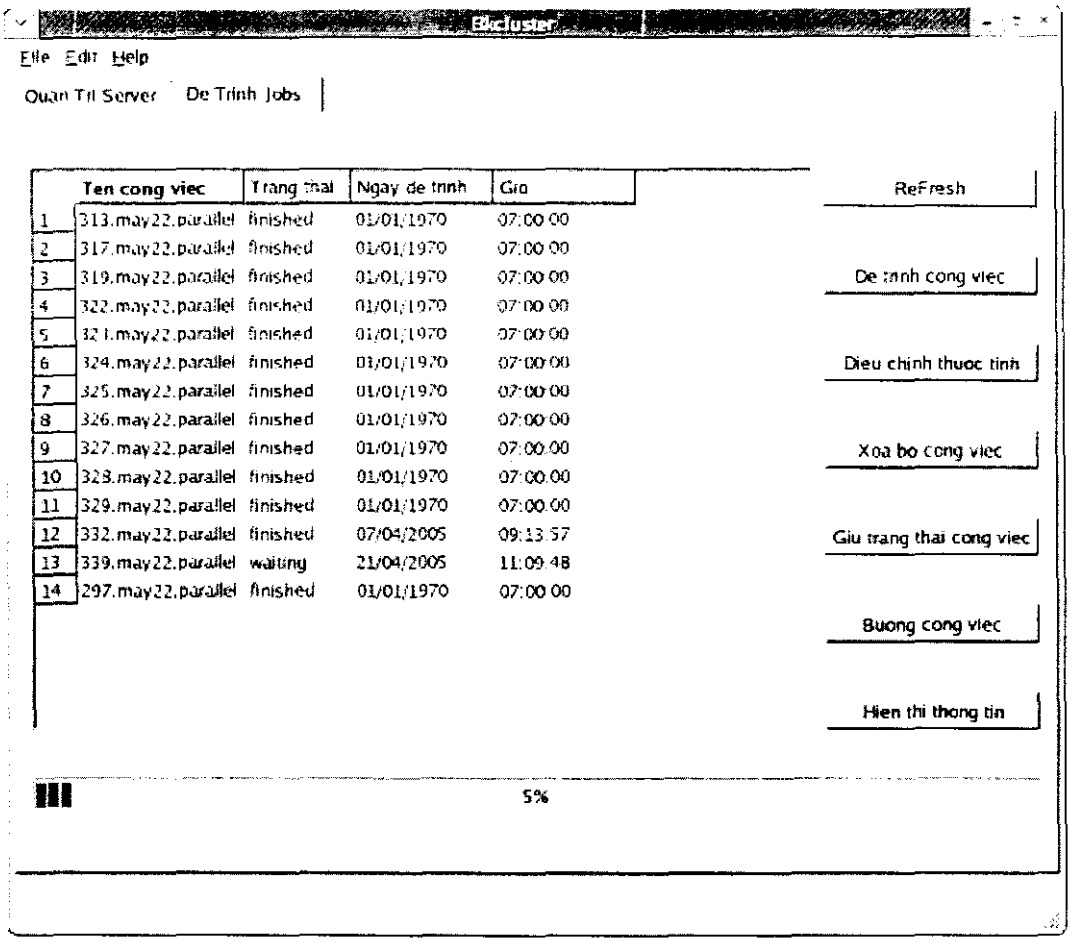
song song một cách dễ dàng (đây là công việc của bộ User Tool).

Cùng với toàn bộ các bạn trong nhóm HPC, sau quá trình thực tập của mình trên trung tâm em đã hoàn thành bộ công cụ theo yêu cầu ban đầu đặt ra và đã cài đặt thử trên trung tâm HPC kết quả là hệ thống chạy ổn định. Tuy nhiên do thời gian và khả năng có hạn nên chương trình còn một số hạn chế nhất định, em hy vọng rằng những hạn chế này sẽ được khắc phục trong thời gian sớm nhất. Sau đây là một số giao diện của hệ thông BKluster và bộ User Tool

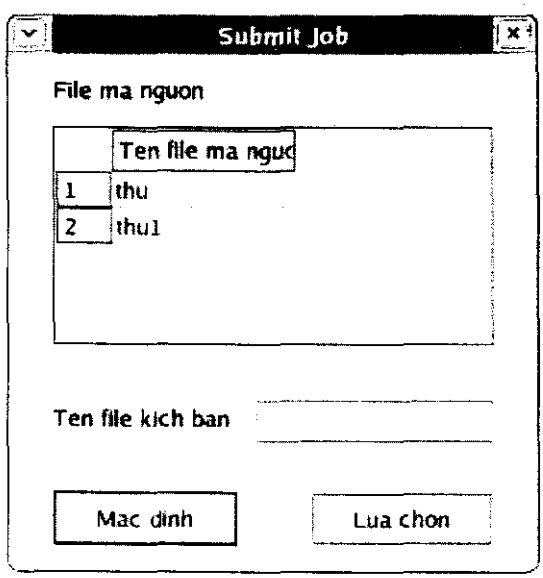


Hình 5-13 Giao diện của toàn bộ hệ thống BKluster

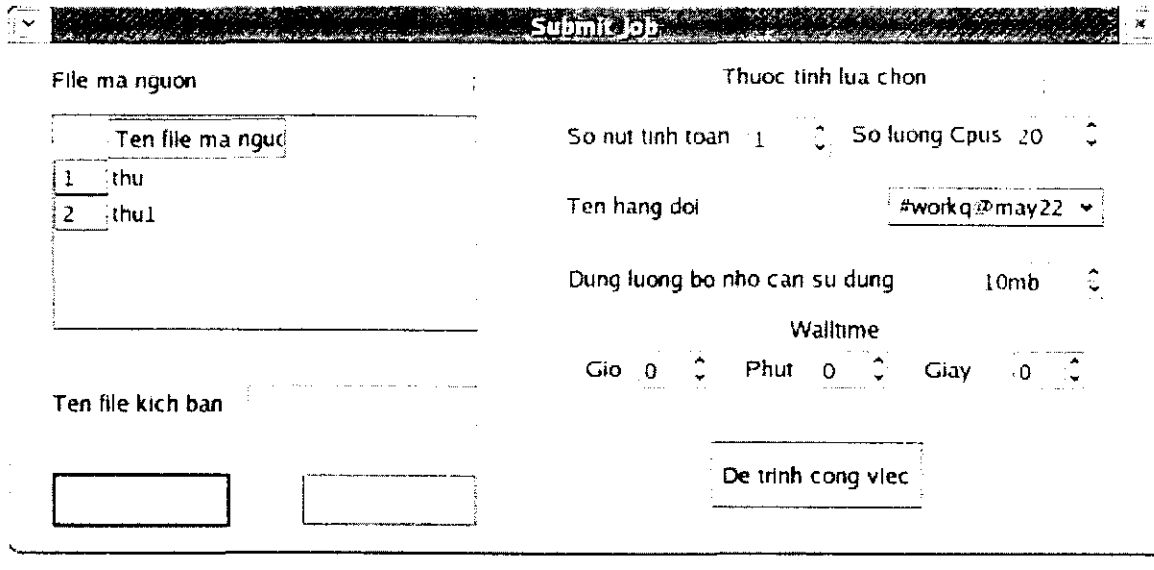
Giao diện chính của User Tool



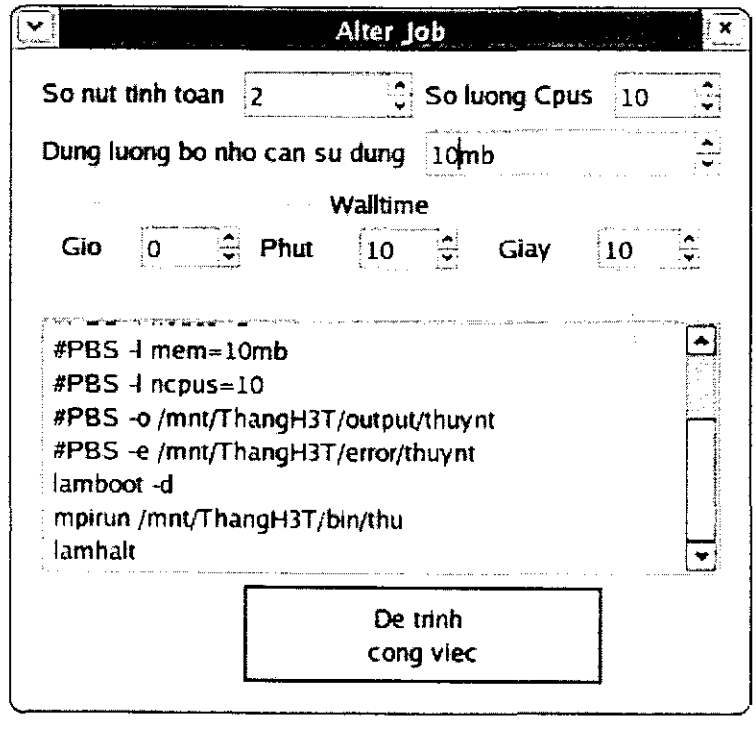
Hình 6. 1 Giao diện của ban đầu User Tool



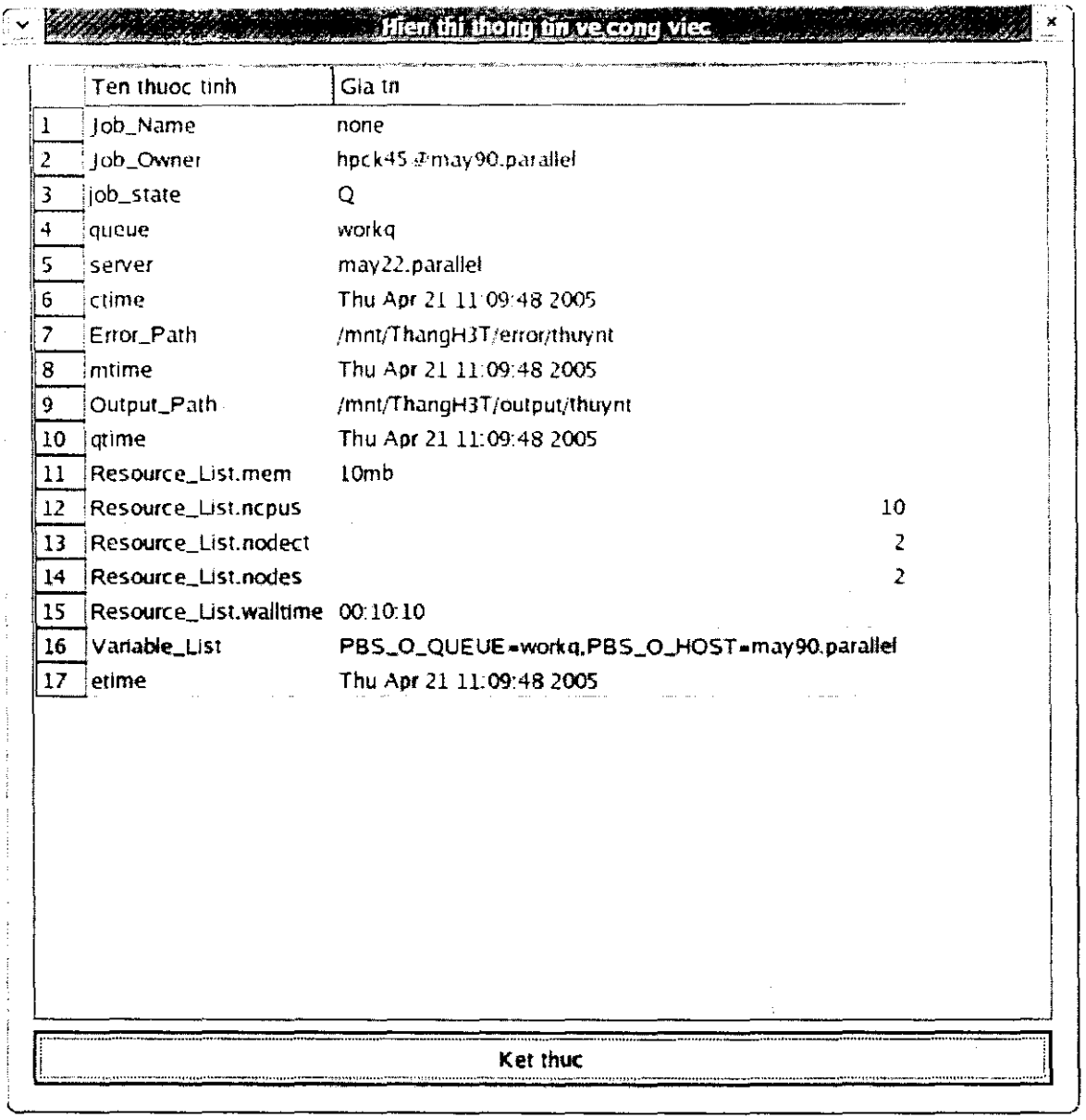
Hình 6. 2 Chức năng đệ trình công việc với các tham số mặc định



Hình 6. 3 Đệ trình công việc với tham số do người dùng lựa chọn



Hình 6. 4 Chức năng điều chỉnh các tham số của một công việc



Hình 6. 5 Chức năng hiển thị thông tin chi về công việc

5.4.2 Định hướng phát triển trong tương lai.

Có thể tích hợp thêm bộ lập lịch Maui vào hệ thống như vậy hệ thống sẽ có nhiều cơ chế lập lịch hơn, thêm nữa nếu có thêm thành phần Maui vào hệ thống sẽ cho phép theo dõi các thông tin bên trong hệ thống một cách trực quan hơn, do Maui cho phép người dùng xem các thông tin về hệ thống dưới dạng các loại biểu đồ và như vậy sẽ cho chúng ta một cái nhìn trực quan hơn về toàn bộ hệ thống.

Ngoài ra do nếu chỉ sử dụng mình hệ thống BKluster thì nhiều khi hệ thống của chúng ta không thệ đáp ứng được các bài toán đặt ra cho dù đã áp dụng tính toán song song rồi và công nghệ tính toán lưới đã ra đời để có thể sử dụng được lượng tài nguyên rộng lớn hơn ở trên mạng, khi đấy mỗi hệ thống Cluster sẽ chỉ là một phần tử trong lưới. Chính vì lợi ích của tính toán lưới nên trong tương lai lâu dài hệ thống BKluster cũng sẽ có kế hoạch để tham gia vào lưới với tư cách là một nút tính toán trong lưới.

CHƯƠNG 6 Module Quản Lý Các Gói Phần Mềm

6.1 Nhiệm vụ của đề tài:

Xây dựng chương trình quản lý các gói phần mềm (BKluster package manger) trong hệ thống cluster. Chương trình này cho phép cài đặt, nâng cấp, xóa và đồng thời cung cấp thông tin thống kê các gói phần mềm đã được cài đặt trong toàn bộ hệ thống.

6.2 Phân tích đề tài:

Để thực hiện đề tài này, có thể sử dụng các cơ chế thực thi lệnh từ xa như rsh, ssh... Tuy nhiên, cách làm này không mềm dẻo, đòi hỏi người sử dụng phải tiến hành đăng nhập trên tất cả các máy thành viên và phải có một kiến thức sử dụng Linux nhất định mới thực hiện được. Cách thứ hai là cài đặt theo mô hình client/server thông qua socket. Mô hình này sẽ gánh phần công việc nặng nhọc về cho người lập trình, nhưng đổi lại, hệ thống sẽ trong suốt với người sử dụng, cho phép người dùng thao tác một cách đơn giản và nhanh chóng mà không cần phải có kiến thức sâu rộng về hệ thống. Chính vì vậy, chúng em đã chọn đi theo cách làm này.

Từ yêu cầu của đề tài, có thể phân tách chương trình thành 2 thành phần chính :

- **BKluster-package-manager** chạy trên máy chủ, cung cấp giao diện tương tác cho người sử dụng. Thành phần này có nhiệm vụ nhận lệnh từ người sử dụng, rồi truyền yêu cầu đến các iserver trên các máy trạm để thực thi.
- **iserver** chạy trên các máy trạm, có vai trò nhận lệnh từ máy chủ, thực thi các yêu cầu tương ứng, rồi truyền kết quả thực hiện lại cho máy chủ hiển thị. iserver phải được thực thi ở chế độ nền, được gọi ngay khi các máy trạm khởi động, sẵn sàng nhận yêu cầu từ máy chủ.

6.3 Tiến hành công việc:

Do các yêu cầu trên, thành phần iserver được viết bằng ngôn ngữ C++, còn thành phần BKluster-package-manager được xây dựng trên ngôn ngữ QT. Khi được 'khởi động', iserver sẽ lắng nghe trên cổng 4242, khi có bất cứ yêu cầu nào được gửi tới qua cổng này, iserver sẽ gọi những lệnh shell của trình vỏ tương ứng để

thực thi. Việc thực hiện lệnh được tiến hành thông qua các đường ống. Với cơ chế này, các thông tin hiển thị tương ứng lên thiết bị hiển thị đầu ra và lỗi chuẩn (stdout, stderr) có thể lấy được một cách dễ dàng để truyền về cho BKluster-package-manager. Mọi thông tin qua lại giữa BKluster-package-manager và các iserver đều ở dưới dạng text, nên việc xử lý không gặp phải nhiều khó khăn.

```
FILE *file = popen(&cmd_line[0], "r");
while (fscanf(file, "%s", line) != EOF)
{
    Response(string(line) + "\n");
}
pclose(file);
```

Đoạn mã trên mở ra một pipe thực thi lệnh shell chứa trong xâu cmd_line, trả về con trỏ file chứa địa chỉ của pipe. Tiếp đến, sử dụng fscanf để lấy kết quả rồi truyền về cho máy chủ với hàm Response.

```
char *buffer = new char[FILE_BUFFER_SIZE];
memset(buffer,0,FILE_BUFFER_SIZE); //zero mem
int n = read(error[READ],buffer,FILE_BUFFER_SIZE-20);
if (n > 0)
{
      Response(string(buffer) + "\n");
}
```

Đoạn mã trên có nhiệm vụ lấy thông tin từ stderr và truyền về cho máy chủ. error là một pipe "kép", với error[WRITE] (WRITE=0) là pipe dùng để ghi, được kết nối với stderr, còn error[READ] (READ = 1) là pipe dùng để đọc dữ liệu.

BKluster-package-manager sử dụng QThread để quản lý việc trao đổi dữ liệu với các máy trạm. Khi nhận yêu cầu từ người dùng, BKluster-package-manager đọc địa chỉ IP của các máy trạm từ file /etc/hosts đưa vào một mảng. Với mỗi IP, BKluster-package-manager mở ra một thread để thực hiện kết nối tới các iserver trên các máy trạm đồng thời quản lý luồng thông tin trao đổi giữa hai chương trình. Khi tiến hành kết nối tới các máy trạm thông qua IP (trên cổng 4242), nếu máy trạm nào không chạy hoặc chương trình iserver trên máy đó không được khởi động, BKluster-package-manager tự động đóng kết nối và thread tương ứng. BKluster-package-manager đợi dữ liệu chuyển đến từ các iserver và hiển thị.

Cài đặt:

Hiện giờ, BKluster package manger đã được đóng gói thành file nén tar với thao tác cài đặt hết sức đơn giản: copy các file chạy vào các thư mục cần thiết, tạo link trong init.d để iserver tự động chạy khi máy trạm khởi động...

Hạn chế:

Trong thời gian có hạn, việc xây dựng BKluster-package-manager mới chỉ dừng lại ở mức cơ bản nhất. Tuy nhiên, cả chương trình vẫn cho phép người dùng thực thi một lệnh shell bất kì trên toàn bộ hệ thống. Việc thực hiện trơn tru, cho người sử dụng một cái nhìn tổng thể về toàn hệ thống, mọi thao tác thực thi lệnh trên các máy đều trong suốt với người dùng.

Việc quản lý các gói phần mềm cài đặt trên hệ thống còn thô sơ, chưa xây dựng được một cơ sở dữ liệu tập trung.

Chưa có hướng quản lý các gói phần mềm được cài đặt trực tiếp từ mã nguồn, do tính đa dạng trong việc cài đặt phụ thuộc rất nhiều vào ý thích của người lập trình.

Phần ứng dụng Master chạy trên máy chủ hệ Beowulf

Nhiệm vụ :

Ứng dụng Master có nhiệm vụ kết nối và quản lí việc cài đặt trên toàn bộ các slave có trong Cluster cũng như khả năng tương tác với một phần nào đó của Cluster (các Slave đã được cài đặt iserver)

Ứng dụng có khả năng sau :

- Gửi lệnh tới tất cả các slave trong cluster
- Có thể tương tác riêng lẻ tới từng slave trong cluster
- Ứng dụng có giao diện , dễ sử dụng với người dùng
- Có khả năng tùy chọn các slave để thực hiện công việc

Giải pháp :

Sử dụng công cụ QT programming hỗ trợ ngôn ngữ lập trình C++

Tận dụng RPM

Thực hiện :

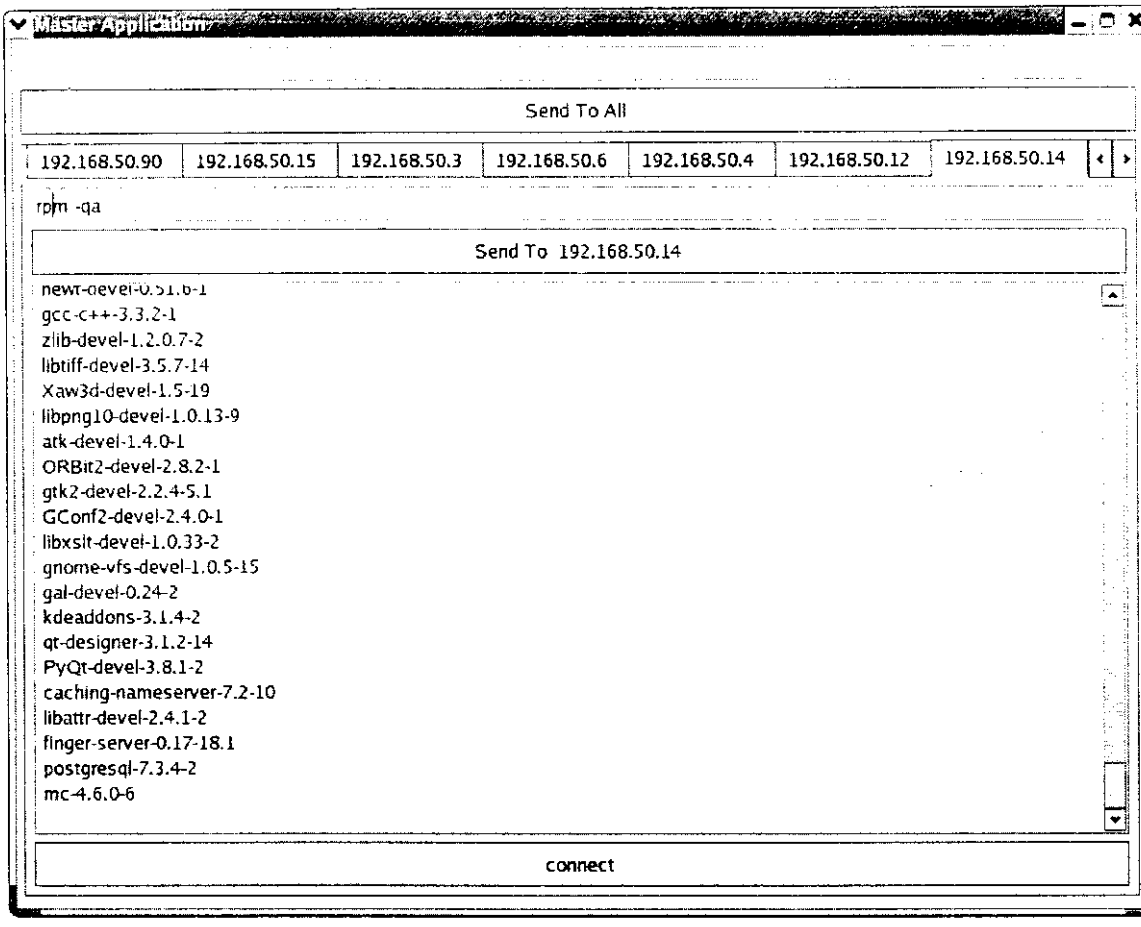
Các Slave có trong Cluster sẽ được lưu địa chỉ trong một file text được đặt tên là hostfile có ghi rõ địa chỉ IP của cac slave đó,mỗi slave trên một dòng .Mỗi khi

cluster mở rộng thay thu hẹp thì chỉ cần thêm vào hoặc xóa đi các dòng tương ứng với các slave mới thêm vào đó (hoặc trong trường hợp chúng ta chỉ muốn cài đặt trên một số máy nhất định thì chúng ta thay đổi file text này để phù hợp với yêu cầu của ta).Điều này có thể thấy ngay trên giao diện của chương trình như trong ảnh chụp màn hình .

Khi thực hiện ,chương trình sẽ đọc file hostfile này , tạo ra các tab tương tác với từng slave , việc sinh ra các tab là hoàn toàn động , phụ thuộc vào số lượng các slave có trong hostfile .Với mỗi slave, ứng dụng Master sẽ tạo một socket kết nối với server đang chạy trên slave đó để đảm bảo khả năng tương tác riêng lẻ với từng slave . Kết quả thực hiện công việc trên slave sẽ được hiện thị trong tab tương ứng với slave đó .Việc lập trình với Socket được QT hỗ trợ với 2 lớp đó là QSocket và QServerSocket .

Với việc tương tác với toàn bộ hệ thống , công việc được tiến hành như sau: mỗi khi muốn gửi lệnh tới toàn bộ slave trong cluster thì ứng dụng sẽ tạo ra các thread , các thread này chụi trách nhiệm gửi lệnh đến toàn bộ các slave có trong hostfile ,(việc lập trình với Thread được hỗ trợ với Qthread),tất nhiên công việc này dựa trên mô hình client-server, kết quả thực hiện công việc của từng slave vẫn sẽ được hiển thị trên tab tương ứng với slave đó .

Giao diện của chương trình như sau :



Bảng 6-1 Giao diện module quản lý các gói phần mềm

6.4 Kết quả đạt được và hướng phát triển

6.4.1 Kết quả đạt được

- Việc tương tác của hệ thống đã được đảm bảo.
- Server chạy ổn định ở chế độ nền trên các slave .
- Đã tiến hành cài đặt thử nghiệm các gói phần mềm trên hệ thống của trung tâm HPCC.

6.4.2 Hướng phát triển

Trong thời gian tới hệ thống sẽ có thêm các phần sau :

- Cơ chế bảo mật : bởi vì đây là chương trình quản lí , quyết định cấu hình của toàn bộ cluster , chính vì thế chỉ có người có quyền hoặc được cấp

quyền quản trị hệ thống mới có quyền sử dụng chương trình này , chương trình sẽ có thêm module đăng nhập khi sử dụng chương trình

- Cơ sở dữ liệu : Để tập trung vào việc quản lý việc cài đặt các gói dữ liệu có trên các Slave , hiện nay hệ thống đã đảm bảo được việc tương tác giữa Master và các Slave xong chưa có cơ sở dữ liệu ghi lại các phiên làm việc
- Module tổng hợp thông tin trả về từ các Slave , báo cáo tổng quan về công việc thực hiện trên toàn bộ hệ thống (ví dụ : thực hiện thành công trên bao nhiêu máy , các máy nào có lỗi, các lỗi cụ thể ra sao)

CHƯƠNG 7 Module Quản Lý Cấu Hình Cluster và Quản Lý Người Dùng

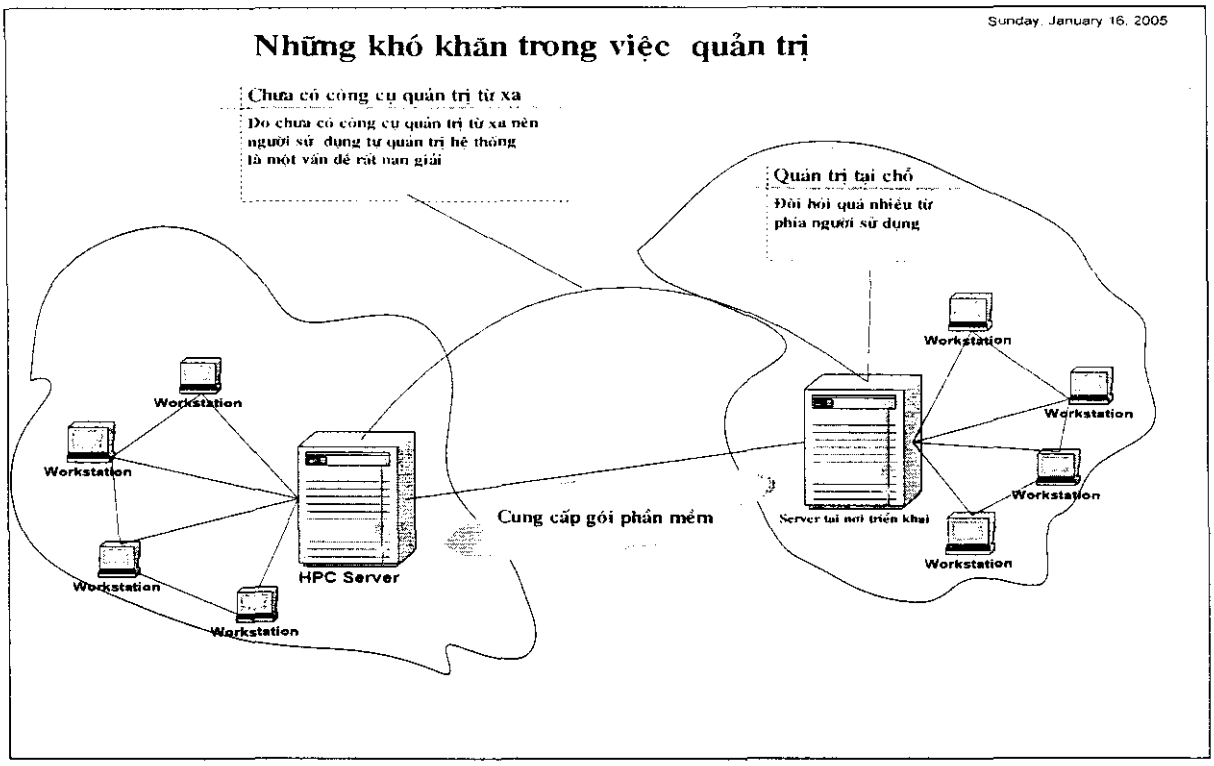
7.1 Phân tích yêu cầu

Trong quá trình phát triển hệ thống tính toán song song tại trung tâm HPC, các đánh giá về góc độ kỹ thuật cho thấy: hệ thống tính toán song song đã đáp ứng được những chức năng cơ bản như lập lịch và thực thi công việc tính toán song song. Tuy nhiên đó là những chức năng cơ bản của một hệ thống tính toán, hệ thống còn nhiều điểm hạn chế nếu nhìn từ góc độ một dịch vụ tính toán.

7.1.1 Các vấn đề đặt ra:

Khó khăn trong việc quản trị hệ thống khi cung cấp dịch vụ dưới dạng đóng gói.

Đối với những người làm tin học ngay cả người có kinh nghiệm trong lĩnh vực tính toán song song phải đối mặt với việc quản trị hệ thống với tập lệnh và các khái niệm phức tạp là một thách thức đáng kể. Nếu triển khai hệ thống tại một trung tâm , mà trung tâm HPCC không thể kết nối trực tiếp tới, việc quản trị hệ thống tính toán hoàn toàn sẽ do người tại trung tâm này đảm nhiệm. Đối với những người nghiên cứu thuộc các lĩnh vực khác như vật lý, hóa học,...việc quản trị một hệ thống tính toán như vậy quả là không dễ dàng.



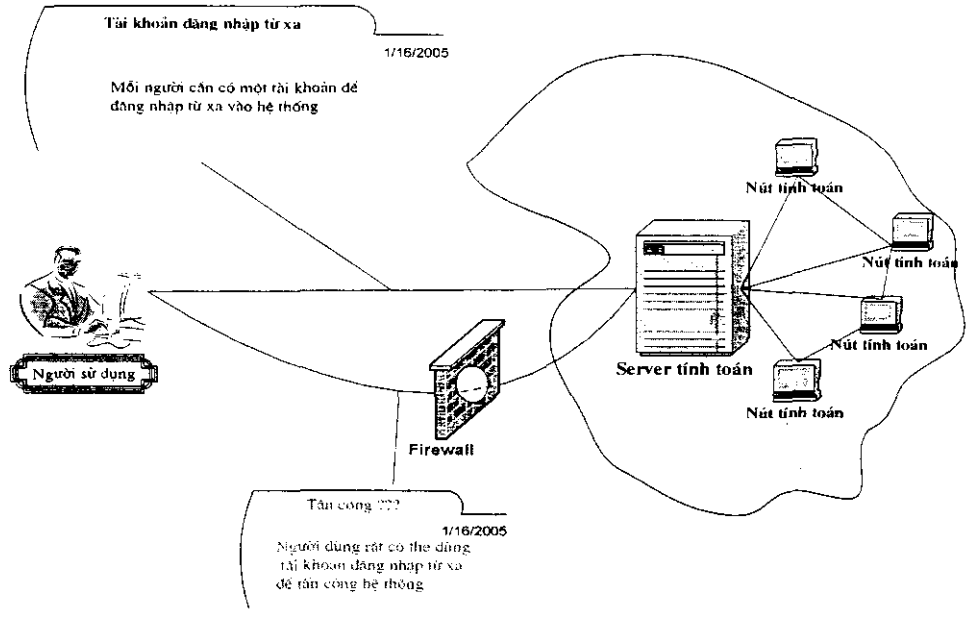
Hình 7-1 Khó khăn trong việc quản trị hệ thống khi cung cấp dạng gói phần mềm

Khó khăn trong việc quản lý người dùng khi cung cấp dịch vụ dưới dạng cấp tài khoản để sử dụng hệ thống.

Dịch vụ cung cấp dưới dạng tài khoản đăng nhập từ xa là dịch vụ cung cấp cho khách hàng một tài khoản của hệ điều hành để người dùng dùng tài khoản đó telnet vào hệ thống và sử dụng công cụ tính toán như thể họ ở tại trung tâm tính toán. Cách làm này là cách cung cấp dịch vụ truyền thống của các hệ thống tính toán song song cũng như tính toán lưới, ưu điểm là đơn giản và dễ dàng điều hành hệ thống. Tuy nhiên, cách cung cấp dịch vụ này có nhược điểm là nó cung cấp cho người dùng một tài khoản trực tiếp có khả năng sử dụng các tài nguyên của hệ thống, vì thế rất khó khăn trong việc bảo mật hệ thống, cũng như theo dõi sự sử dụng tài nguyên.

Sunday, April 24, 2005

Khó khăn trong việc quản trị người dùng



Hình 7-2 Khó khăn trong việc quản trị khi cung cấp dạng tài khoản đăng nhập từ xa

7.1.2 Cách giải quyết:

"Xây dựng công cụ quản trị cấu hình hệ thống tính toán song song" với các chức năng sau:

- Tạo các tài khoản và quản lý người dùng tách biệt với các tài khoản người dùng của hệ điều hành.
- Công cụ cấu hình hệ thống cho phép người quản trị có thể cấu hình hệ thống tại chỗ hoặc từ xa với những hiểu biết cơ bản về hệ thống tính toán song song ghép cụm.
- Công cụ giám sát hệ thống, cho người quản trị có thể theo dõi hoạt động của hệ thống qua các thông tin ghi lại trong quá trình hoạt động của hệ thống, thông tin có thể dưới dạng thông báo bình thường hoặc thông báo lỗi.

7.2 PHÂN TÍCH CÁC YÊU CẦU QUẢN TRỊ

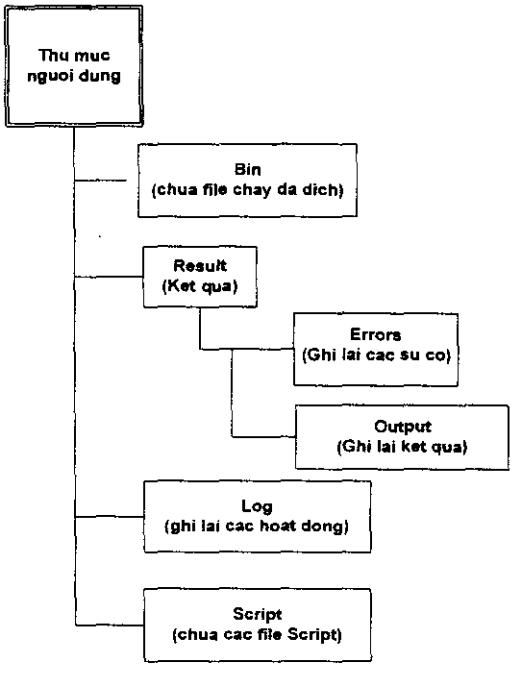
Từ những yêu cầu thực tế của các dịch vụ tính toán và những phân tích ban đầu về việc quản trị dịch vụ tính toán, chương này sẽ đưa ra những yêu cầu cơ bản cần được đảm bảo trong khi xây dựng công cụ quản trị hệ thống. Sau khi đưa ra được những yêu cầu chính của công cụ quản trị, chúng ta sẽ cùng đi sâu phân tích chi tiết các chức năng cần có để đảm bảo các yêu cầu đó.

7.2.1 Các chức năng chính

7.2.1.1 Quản lý người dùng

Mỗi người dùng từ xa muốn sử dụng dịch vụ tính toán phải đăng ký một tài khoản, tài khoản có quyền quản trị sẽ đảm nhận việc chấp nhận đăng ký, phân quyền và quản lý các tài khoản khác.

Mỗi tài khoản người dùng sẽ có một thư mục trên server để lưu các file chạy, script, log file và kết quả việc chạy chương trình.



Hình 7-3 Cấu trúc cây thư mục người dùng

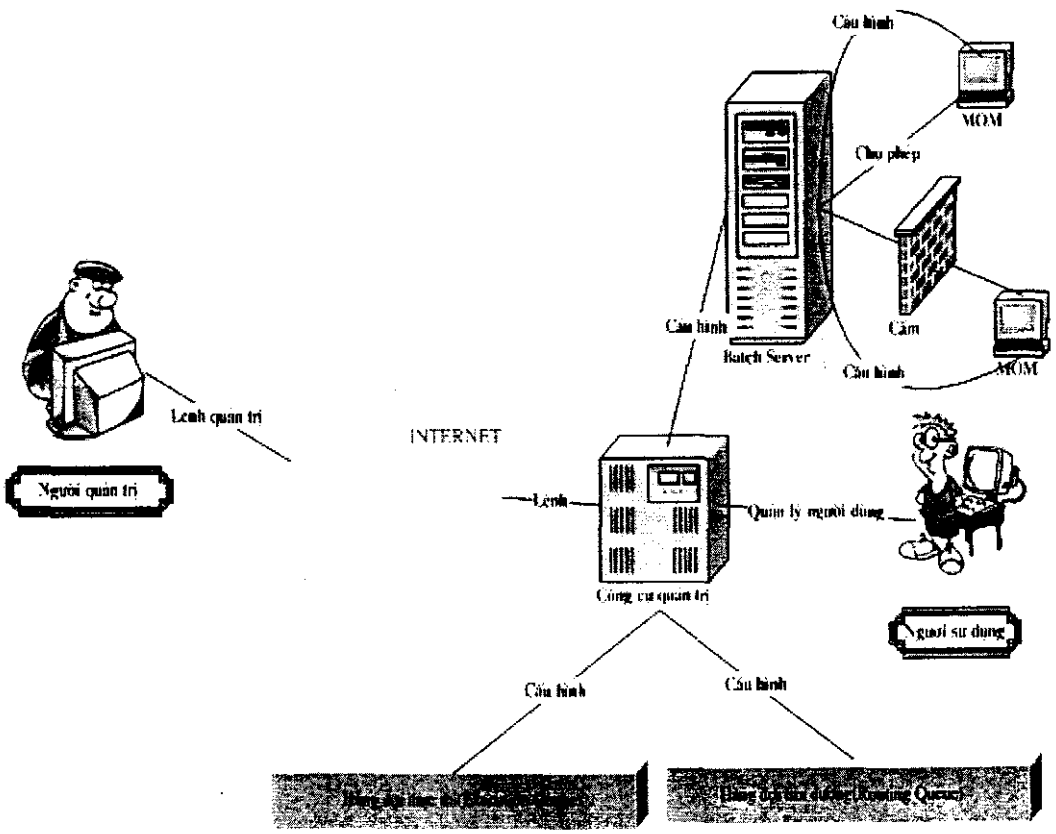
- Thư mục Bin là thư mục lưu lại các file đã dịch của người dùng, tạo ra do

công cụ IDE.

- Thư mục Result lưu lại các thông tin của quá trình chạy các chương trình song song, như lỗi (thư mục Errors), kết quả (thư mục Output), thông tin về quá trình chạy (thư mục Log).
- Thư mục Log lưu lại các thông tin về công việc song song của người dùng như: thời điểm bắt đầu và kết thúc chạy công việc song song,...
- Thư mục Script dùng để lưu các PBS script file do User Tool giúp người dùng tạo ra để chạy các chương trình song song, trong đó có các chỉ dẫn về tài nguyên và file chạy cho PBS.

7.2.1.2 Chức năng cấu hình

Chức năng cấu hình cung cấp phương tiện cho người quản trị thao tác với các nút tính toán, server tính toán, và các hàng đợi công việc, cũng như thiết lập các thuộc tính khác như cấp quyền quyền, cho phép/cấm các nút khác tham gia vào hệ thống...



Hình 7-4 Chức năng cấu hình của công cụ quản trị

7.2.1.3 Chức năng giám sát

Chức năng giám sát thực hiện việc lưu lại thông tin về tình trạng về server, hàng đợi công việc và các công việc đã thực hiện và kết quả của chúng sau chu kỳ thời gian nhất định, việc lưu lại các thông tin này nhằm mục đích thống kê, tối ưu hóa và khắc phục những lỗi nghiêm trọng có thể xảy ra. Trong hệ thống

Chức năng giám sát còn cung cấp phương tiện giúp người quản trị theo dõi trạng thái các nút tính toán tại thời điểm hiện tại (nút nào đang chạy, nút nào bị down, nút nào free...) ngoài ra công cụ còn có tham vọng theo dõi các tiến trình của một job chạy trên các nút tính toán trong hệ thống tính toán

7.2.2 Chi tiết các yêu cầu quản trị.

Trong phần này chúng ta sẽ phân tích kỹ về các yêu cầu cụ thể đảm bảo các chức năng chính của công cụ quản trị.

7.2.2.1 Quản trị người dùng

Để quản trị người dùng cần các chức năng sau:

- Thêm người dùng vào hệ thống: thêm người dùng vào hệ thống bao gồm việc thao tác với thư mục của người dùng, cập nhật cơ sở dữ liệu...
- Lấy thông tin về một người dùng.
- Cập nhật thông tin của người dùng: chẳng hạn như thêm quyền cho người dùng đó. Trong BKluster người dùng được chia làm ba cấp: người quản trị là người có toàn quyền đối với hệ thống, thành viên của hệ thống là người có thể thực hiện các công việc song song, người dùng thông thường là người chỉ có một số quyền hữu hạn như xem thông tin về hệ thống.
- Xóa người dùng

7.2.2.2 Quản lý nút trong trong BKluster

Ở mức độ quản lý việc phân tải Công cụ quản trị cần thực hiện các chức năng sau:

- Xem trạng thái của các nút tính toán(pbs_mom), các trạng thái có thể là tắt, không tham gia tính toán, hoặc đang phục vụ tính toán... .
- Thêm một nút tính toán vào hệ thống
- Bớt một nút ra khỏi hệ thống tính toán
- Thay đổi cấu hình của nút tính toán tham gia vào hệ thống.

7.2.2.3 Quản trị PBS Server

PBS server cần có các chức năng quản trị sau đây:

- Lấy thông tin về trạng thái của server: chúng ta có thể lấy được thông tin về bash server như có bao nhiêu hàng đợi, cho phép cấm người dùng, nút tính toán tham gia vào hệ thống hay không...
- Tắt PBS server: trong một số trường hợp cần thiết cần phải tắt PBS server cho mục đích bảo trì hay an ninh của hệ thống...
- Khởi động lại server tính toán.
- Quản lý hàng đợi công việc
- Để quản lý các hàng đợi công việc chúng ta cần các chức năng sau đây:
- Lấy thông tin về các hàng đợi công việc.
- Tạo hàng đợi: tạo các hàng đợi trên server với các thuộc tính do người quản trị cung cấp khi tạo hàng đợi
- Thay đổi các thuộc tính cho các hàng đợi: có thể thêm các thuộc tính, chỉnh sửa tài nguyên... cho hàng đợi.
- Thay đổi trạng thái của hàng đợi (chạy, dừng...)
- Xóa hàng đợi.
- Lưu các thông tin về hoạt động của hệ thống

Các thông tin tập trung vào ba loại : thứ nhất, thông tin về các hoạt động của người dùng (tổng số công việc, thời gian thực hiện, thời gian yêu cầu...) ; thứ hai, thông tin ghi lại các hoạt động của server; thứ ba, thông tin về hoạt động của các hàng đợi (độ hiệu quả, số lượng công việc ....).

Lưu thông tin về hoạt động của người dùng, xử lý với mục đích thống kê

Lưu thông tin về hoạt động của server

Lưu thông tin về hoạt động của các hàng đợi.

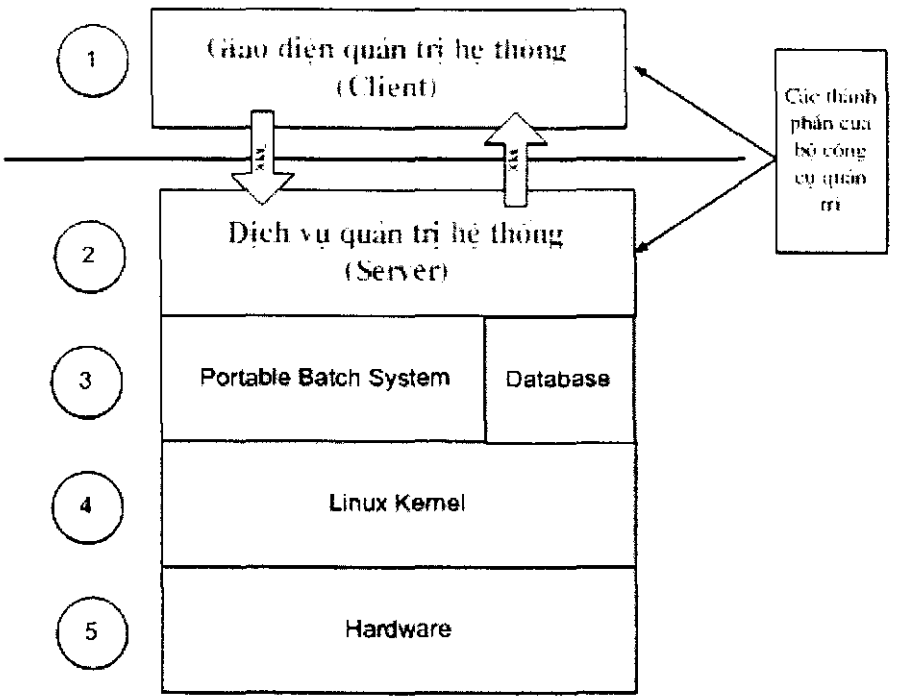
7.3 KIẾN TRÚC BỘ CÔNG CỤ QUẢN TRỊ

Xây dựng kiến trúc cho một phần mềm là một việc khó, đòi hỏi người phát triển phải dựa vào kiến trúc vật lý, những yêu cầu chức năng của hệ thống và những yêu

cầu cho sự phát triển lâu dài của hệ thống. Dưới đây là một đề xuất về kiến trúc của công cụ quản trị.

7.3.1 Bộ công cụ quản trị trong kiến trúc chung của BKluster

Bộ công cụ quản trị có chức năng chính là cung cấp công cụ cho người quản trị cấu hình và quản lý hệ thống từ xa, và tuân theo kiến trúc chung của BKluster nên kiến trúc được lựa chọn là kiến trúc client-server.



Hình 7-5 Bộ công cụ quản trị hệ thống trong kiến trúc chung của BKluster

- **Thành phần giao diện quản trị hệ thống**: là thành phần giao tiếp với người quản trị hệ thống, cung cấp giao diện cho người quản trị thao tác với hệ thống.
- **Thành phần dịch vụ quản trị hệ thống**: là thành phần xử lý các yêu cầu gửi tới từ xa và chuyển các yêu cầu đó thành các yêu cầu quản trị đối với hệ thống PBS, và cơ sở dữ liệu người dùng, sau đó nhận các phản hồi và gửi trả về phía giao diện quản trị hệ thống.
- **Thành phần Portable Batch System**: là hệ thống quản lý tài nguyên và phân tải, thành phần phần mềm cốt lõi của hệ thống tính toán.
- **Thành phần User Database**: là cơ sở dữ liệu người dùng của hệ thống tính toán.

- **Thành phần Linux Kernel**: là nhân hệ điều hành Linux cung cấp môi trường cho các ứng dụng phía trên.
- **Thành phần Hardware**: là các máy tính cá nhân cấu hình trung bình được kết nối tạo ra hệ thống tính toán phân cụm.

Trên đây là kiến trúc chung của công cụ quản trị dưới góc nhìn tổng quan toàn bộ hệ thống tính toán. Bộ công cụ quản trị gồm hai thành phần là thành phần thứ nhất giao diện quản trị hệ thống (client) và thứ hai dịch vụ quản trị hệ thống (server), các client kết nối với server qua các socket và gửi tới đó các yêu cầu quản trị từ xa, server xử lý các yêu cầu này chuyển chúng thành các lệnh tương ứng với các chức năng quản trị như quản trị người dùng, cấu hình hệ thống hay giám sát hoạt động của hệ thống...server sau khi chuyển các yêu cầu tới PBS và cơ sở dữ liệu người dùng, lấy lại các kết quả cần thiết và gửi trả lại phía client. Việc truyền nhận dữ liệu giữa client và server sử dụng XML.

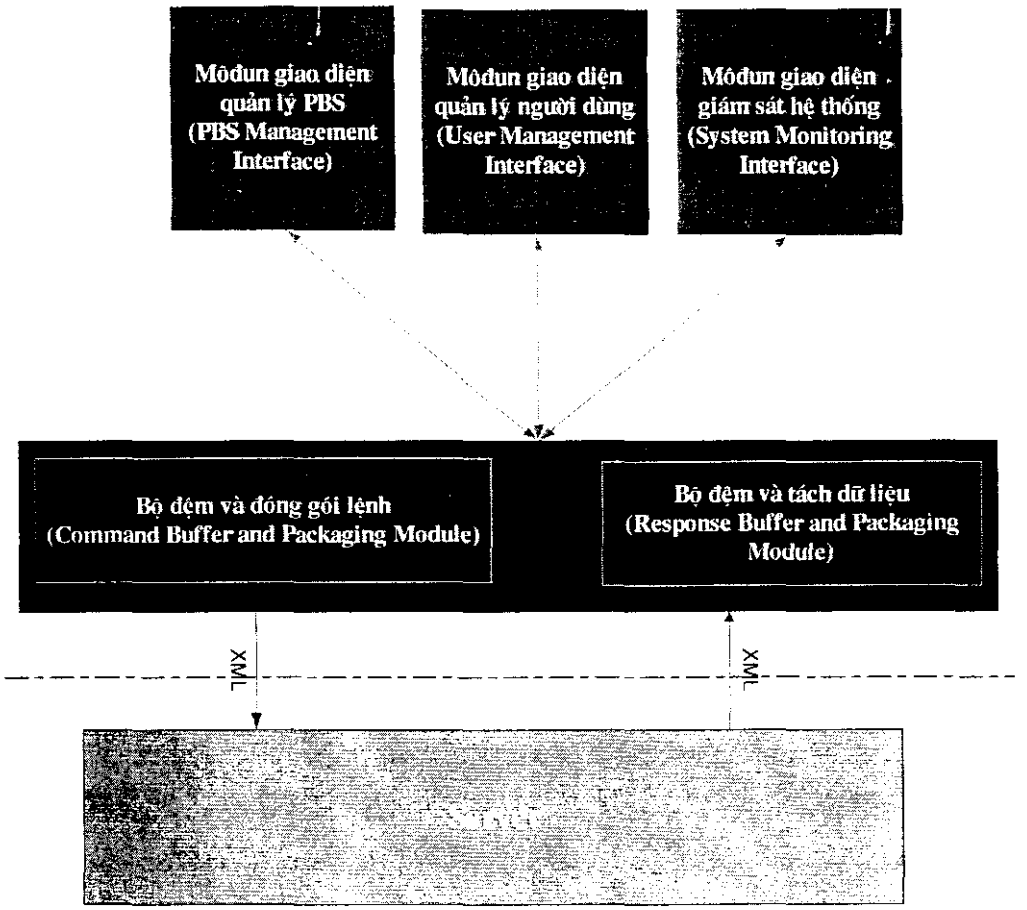
7.3.2 Các module trong bộ công cụ quản trị hệ thống

Để làm rõ hơn kiến trúc của bộ công cụ quản trị hệ thống, báo cáo sẽ trình bày các module trong hai phần chính là giao diện quản trị hệ thống (client) và dịch vụ quản trị hệ thống (server).

7.3.2.1 Kiến trúc Client

Client có ba nhiệm vụ chính :

- Nhận lệnh từ phía người quản trị đóng gói, chuyển tới server.
- Các lệnh sinh ra từ các thao tác của người sử dụng sau đó được các module giao diện chuyển tới bộ đệm và đóng gói lệnh, các lệnh sẽ được đóng gói thành các xâu XML và chuyển tới phía server, nhận phản hồi từ phía server, tách dữ liệu Bộ đệm và tách dữ liệu sẽ nhận phản hồi từ phía server dưới dạng XML và tách dữ liệu ra khỏi định dạng XML, các dữ liệu tách ra sẽ đưa tới các module giao diện để hiển thị.
- Trình bày dữ liệu : Dữ liệu sau khi được bóc tách sẽ được các module giao diện trình bày để người quản trị có thể biết các thông tin về hệ thống và những thao tác của họ đã thay đổi cấu hình hệ thống ra sao.



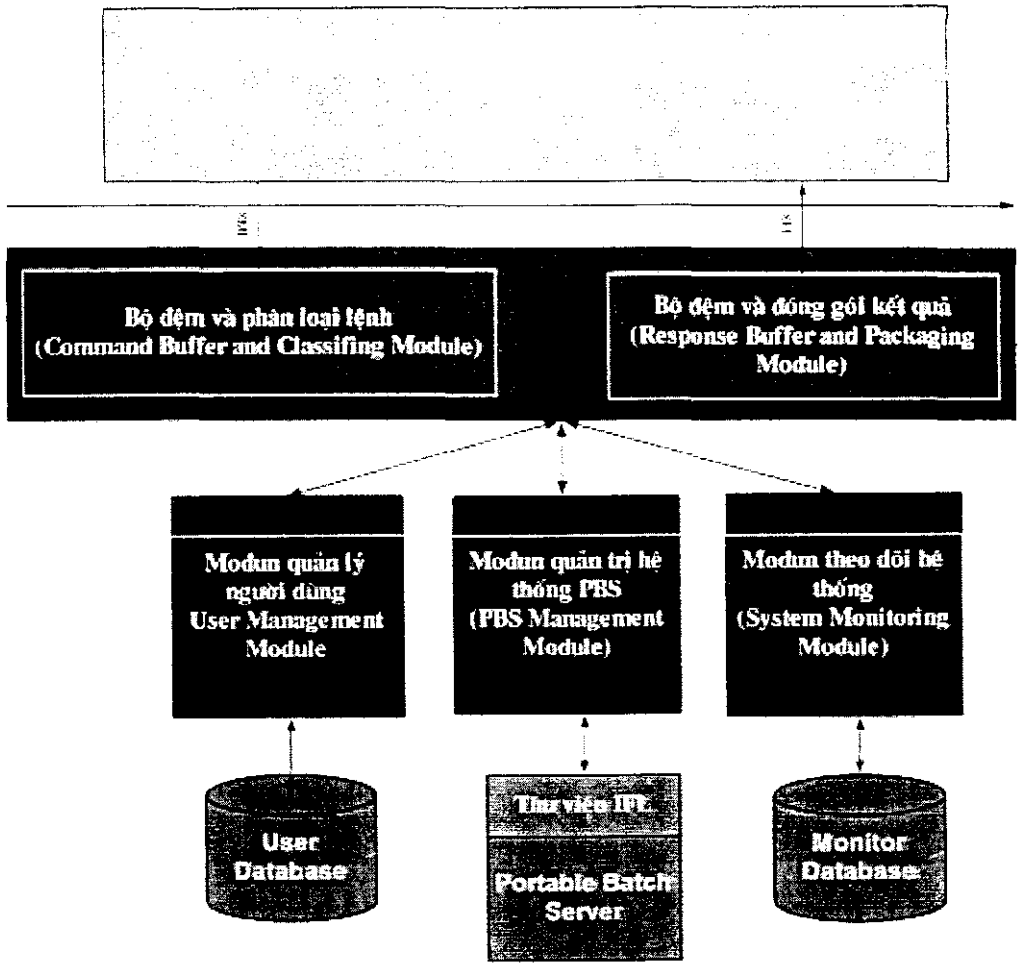
Hình 7-6 Các thành phần phía Client

7.3.2.2 Kiến trúc Server

Khi có kết nối từ client, server sẽ tạo tiến trình con phục vụ và tải các module cần thiết, khi có lệnh gửi tới bộ đệm (Command Buffer and Classifying module) sẽ bóc tách dữ liệu dưới dạng XML, xem lệnh đó thuộc loại lệnh nào và gửi tới các module xử lý tương ứng. Có các loại lệnh như sau

Lệnh lấy thông tin

Lệnh thao tác (thêm, bớt, xóa, sửa... các thuộc tính của PBS, hay cơ sở dữ liệu người dùng...).



Hình 7-7 Các thành phần của server

- Nếu lệnh đó là lệnh thao tác với PBS, bộ xử lý (PBS Management Module) chuyển lệnh dưới dạng XML thành các batch struct để thao tác với PBS thông qua thư viện IFL.
- Nếu lệnh đó là lệnh thao tác quản lý người dùng thì module xử lý(User Management Module) sẽ chuyển thành các thao tác truy xuất cơ sở dữ liệu và lấy các thông tin cần thiết.
- Nếu lệnh đó là lệnh yêu cầu tập hợp dữ liệu để thực hiện thống kê thì lệnh được chuyển cho module theo dõi hệ thống xử lý.

Các kết quả trả về được các module xử lý tương ứng chuyển tới bộ đệm và được đóng gói dưới dạng XML, gửi tới các client. Kết quả trả về có thể ở dưới các dạng sau:

- Thông báo đã thực hiện lệnh
- Thông báo lỗi khi thực hiện lệnh

Dữ liệu, khi lệnh gửi tới đòi hỏi các dữ liệu để hiển thị phía Client

Trên đây là tổng quan về kiến trúc của bộ công cụ quản trị hệ thống, những chi tiết về việc đóng gói dữ liệu, định dạng XML của các lệnh, và kết quả trả về sẽ nói rõ hơn ở chương tiếp theo khi tiến hành xây dựng hệ thống.

7.4 XÂY DỰNG CÔNG CỤ QUẢN TRỊ

Phần này sẽ trình bày chi tiết về các khái niệm liên quan tới việc quản trị hệ thống và hoạt động của hệ thống, đưa ra giao thức truyền nhận dữ liệu của hai phía client từ xa và sever quản trị từ đó đưa ra một số lớp và cấu trúc dữ liệu cần thiết để xây dựng công cụ quản trị hệ thống.

7.4.1 Các khái niệm cơ sở

7.4.1.1 Phân lớp người dùng

Một trong những nhiệm vụ chính của công cụ quản trị là quản lý người dùng tách biệt khỏi hệ điều hành để tránh các nguy cơ an toàn thông tin (quản lý người dùng chung với cơ sở dữ liệu người dùng của hệ điều hành là cách quản lý của hầu hết các hệ thống tính toán song song ghép cụm), để đảm bảo quản lý người dùng hiệu quả người dùng hệ thống được phân ra làm ba lớp người dùng với những quyền tương ứng

Người dùng trong hệ thống	Quyền của người dùng
Người quản trị	- Toàn quyền với hệ thống và các người dùng khác -Cấu hình, thêm bớt, xóa, sửa tất cả các đối tượng thuộc hệ thống phân cụm bao gồm: hàng đợi công việc, server tính toán, nút tính toán, các công việc.
Thành viên	- Cần đăng ký trước khi trở thành thành viên của hệ thống - Có quyền xem cấu hình hệ thống

	- Đệ trình, thực thi công việc - Toàn quyền với thư mục được cung cấp trên server quản trị
Khách	- Chỉ có quyền xem cấu hình hệ thống, không có quyền nào khác.

Bảng 7-1 Phân lớp người dùng

7.4.1.2 Hệ thống tính toán phân cụm - các khái niệm cơ sở

Đây là những khái niệm cơ bản của các hệ thống tính toán song song ghép cụm nói chung và của hệ thống PBS nói riêng. Dưới đây là trình bày một cách tóm tắt các khái niệm đó, chi tiết của nó có thể xem trong tài liệu tham khảo [PBS External Reference Specification]

Nút tính toán (Node)

Một máy tính với một hệ thống xử lí riêng, một bộ nhớ độc lập, một hoặc nhiều bộ vi xử lí và một địa chỉ mạng được gọi là một nút tính toán.Thông thường người ta thường dùng Excution Host để ám chỉ đến các nút tính toán. Ví dụ một máy tính như SGI Origin 2000 thường có nhiều bộ vi xử lý được chạy dưới một hệ điều hành duy nhất và được coi như là một nút đối với PBS. Một hệ thống như IBM SP chứa rất nhiều đơn vị xử lý, mỗi đơn vị xử lý đều chạy dưới một hệ điều hành riêng và hệ thống này này là sự tập hợp của rất nhiều nút.

Các thuộc tính của một nút mà ta thường quan tâm được cho trong bảng sau:

Các thuộc tính	Ý nghĩa
Node_state	Trạng thái của nút, có thể nhận một hoặc kết hợp các các giá trị sau D – node đang bị down F – node đang rỗi (free) O – node đang ở trạng thái offline U – node Unknown R – node reserved

	E – node đang chạy một job ở exclusive – mode
	S – node đang chạy một job ở share – mode
	B – node đang bận
Node_type	Kiểu của một nút, ở đây ta phân biệt hai kiểu node :
	Timeshare – node ở dạng timeshare, tức nó chỉ cho phép hệ thống của ta sử dụng nó ở dạng chia sẻ thời gian, chứ nó không trực tiếp là một bộ phận của hệ thống cluster tính toán.
	Cluster – node này là một thành viên của hệ thống cluster tính toán.
Name	Tên của nút.
Propertice	Thuộc tính của nút.
Jobs	Tên của các job đang chạy trên nút đó tại thời điểm hiện tại
Ncpus	Số lượng các CPU.
Physmem	Tổng bộ nhớ vật lý của nút
Loadave	Tải trung bình

Bảng 7-2 Các thuộc tính của Nút tính toán

Hàng đợi công việc (Queue)

Hàng đợi là một đối dạng đối tượng được quản lí bởi Server. Một hàng đợi có thể bao gồm nhiều công việc(Job), tập các thuộc tính và một tập hợp các đối tượng dữ liệu bên trong. Các công việc được định vị trong các hàng đợi và trở thành các phần tử của hàng đợi. Các máy khách có thể lấy về các thông tin của hàng đợi và các công việc bên trong hàng đợi thông qua các yêu cầu tới Server.

Có hai dạng hàng đợi cơ bản:hàng đợi định tuyến và hàng đợi thực thi. Và các thuộc tính bên trong của hàng đợi quyết định kiểu loại của mỗi hàng đợi.

Khi một công việc nằm bên trong hàng đợi định tuyến, nó có xu hướng được chuyển tới một vị trí khác và khái niệm vị trí ở đây có thể bao gồm các hàng đợi khác cùng nằm trên Server hoặc các hàng đợi của các Server khác.

Một Công việc được coi là đã được lấy ra khỏi hàng đợi định tuyến khi:

- Công việc đã thực sự được định vị thành công tới một hàng đợi khác.
- Công việc đã bị xoá khỏi hàng đợi.
- Công việc đã bị người sử dụng chuyển tới một hàng đợi khác.
- Công việc đã bị bỏ qua bởi Server

Khi một công việc nằm trong hàng đợi thực thi, nó đang nằm trong tình trạng chuẩn bị được thực hiện. Một công việc sẽ vẫn là phần tử của hàng đợi thực thi kể cả khi nó đã được lựa chọn để đưa vào thực hiện. Và một công việc được coi là đã được lấy ra khỏi hàng đợi thực hiện khi:

- Công việc đã được thực hiện và đã kết thúc.
- Công việc đã được chuyển tới một hàng đợi khác.
- Công việc đã bị người sử dụng bỏ qua không thực hiện.

Dưới đây là bảng các thuộc tính công khai của hàng đợi , người sử dụng hệ thống có thể đặt lại giá trị cho các thuộc tính này nếu được phép:

Các thuộc tính	Ý nghĩa
Acl_Group_Enable	Nếu thuộc tính này được gán bằng True, Server sẽ sử dụng acl_group với mục đích như một danh sách về kiểm soát quyền sử dụng Queue của các nhóm người sử dụng. Định dạng giá trị của thuộc tính này là Boolean.
Acl_groups	Danh sách này cho phép hoặc cấm các thành viên của nhóm được đưa ra trong danh sách có thể đưa các Job vào Queue. Đây là danh sách các nhóm truy nhập của Server không phải là các nhóm sử dụng của các máy đệ trình Job. Định dạng của thuộc tính:[+|-group_name[,...]] và giá trị mặc định của thuộc tính này là tất cả các nhóm sử dụng đều được quyền sử dụng Queue.
Acl_host_enable	Khi thuộc tính này được đặt bằng True, Server sẽ sử dụng thuộc tính acl_hosts.

Acl_hosts	Danh sách của các máy khách có thể đệ trình Job tới Queue. Định dạng[+|-hostname[....]]
Acl_user_enable	Khi thuộc tính này được đặt bằng True Server sẽ sử dụng thuộc tính acl_users làm danh sách kiểm soát truy nhập đối với Queue. Định dạng: Boolean.
Acl_users	Danh sách người sử dụng được phép hoặc không được phép đệ trình Job trên Queue. Định dạng [+|-user[@host][,...]]
Enabled	Thuộc tính này được đặt là True hoặc False sẽ quyết định việc Queue có chấp nhận một Job mới vào trong hàng đợi hay không. Định dạng: Boolean.
Max_queuable	Số lượng cực đại các Job có thể cho phép trong Queue tại một thời điểm.
Max_running	Số lượng tối đa các Job trong hàng đợi có thể được đưa vào trạng thái thực hiện hoặc được định đường lại tại một thời điểm.
Priority	Mức độ ưu tiên của một Queue so với các Queue khác có cùng loại trong hệ thống. Trong hệ thống tồn tại hai dạng hàng đợi là hàng đợi thực hiện và hàng đợi định đường.
Resources_max	Số lượng tối đa của mối loại tài nguyên mà có thể được phép yêu cầu của các Job trong hàng đợi.
Resources_min	Số lượng nhỏ nhất mỗi loại tài nguyên mà mỗi một Job trong hàng đợi có thể yêu cầu.
Resources_default	Danh sách các giá trị mặc định của tài nguyên được đặt giới hạn cho một Job trong hàng đợi và các giá trị này được gán cho các Job mà các tài nguyên này không được đặt giá trị.

Started	Nếu thuộc tính được gán bằng True, các Job trong hàng đợi có thể được đưa vào trạng thái lập lịch để thực hiện và khi được gán giá trị là False, Queue bị dừng tạm thời.
Checkpoint_min	Xác định khoảng thời gian nhỏ nhất cho phép thực hiện kiểm tra định kì đối với một Job trong Queue.
Resources_available	Danh sách các tài nguyên và số lượng của các tài nguyên này cho phép sử dụng đối với tất cả các Job trong hàng đợi. Tổng số mỗi loại tài nguyên được sử dụng bởi tất cả các Job không được vượt quá số lượng này.
Max_user_run	Số lượng tối đa các Job trong hàng đợi thuộc một người sử dụng có thể được tại một thời điểm xác định.
Max_group_run	Số lượng tối đa các Job trong hàng đợi thuộc về bất cứ một người sử dụng trong nhóm có thể được chạy tại một thời điểm.

Bảng 7-3 Các thuộc tính của hàng đợi công việc

Server tính toán

Đối tượng máy chủ hay thành phần quản lí tiến trình (PBS Server) . Đây là thành phần phức tạp nhất của hệ thống. Máy chủ có nhiệm vụ quản lý các thông tin chung của hệ thống, lắng nghe yêu cầu của người dùng hoặc của các module khác, xử lý và trả lại kết quả tương ứng.Ý nghĩa và tên gọi các thuộc tính của máy chủ (Server) tương tự như các thuộc tính của hàng đợi(Queue).

Tài nguyên (Resource)

Trong hệ thống PBS các tài nguyên hệ thống là đại diện cho những gì mà một công việc (Job) yêu cầu và sử dụng trong quá trình thực hiện. Cụ thể hơn nữa tài nguyên hệ thống là những gì mà các công việc cần để thực hiện như các nút tính toán, các gói phần mềm hoặc là những gì mà công việc sẽ sử dụng trong quá trình chạy như thời gian CPU, bộ nhớ thực hoặc bộ nhớ ảo. Các tài nguyên được sử

dụng trong PBS chúng ta có thể phân làm hai loại khác nhau: loại thứ nhất là các tài nguyên xác định một công việc có thể được đưa vào một hàng đợi nào đó hay không, và dạng thứ hai là các tài nguyên giới hạn số lượng tài nguyên một công việc có thể sử dụng trong quá trình thực hiện.

Nhìn chung khi một công việc được đệ trình để thực hiện, chúng sẽ được đưa vào các hàng đợi định tuyến(Routing Queue). Sau đó các tài nguyên được yêu cầu bởi bởi các công việc sẽ được so sánh với tài nguyên của các hàng đợi thực hiện đối với hàng đợi định tuyến để xác định xem công việc đó sẽ được đưa vào hàng đợi nào. Thông thường công việc đó sẽ được đưa vào hàng đợi đầu tiên thoả mãn tiêu chí: tất cả các tài nguyên hệ thống được yêu cầu bởi công việc đó đều nằm trong giới hạn tài nguyên nhỏ nhất và lớn nhất của hàng đợi qui định. Nếu một công việc khi được đệ trình lên hệ thống không xác định các tài nguyên yêu cầu một cách tường minh các tài nguyên này sẽ không được giới hạn. Nhưng khi một công việc đã được đưa vào trong một hàng đợi nào đó, các giá trị của các tài nguyên không được xác định này có thể được gán bởi các giá trị tài nguyên mặc định của hàng đợi, hoặc giá trị mặc định của tài nguyên xác định cho thành phần quản lí tiến trình, hoặc giá trị cực đại của tài nguyên cho phép của hàng đợi, hoặc giá trị cực đại của tài nguyên của thành phần quản lí tiến trình theo thứ tự được nêu trên tuỳ thuộc vào các giá trị được nêu trước có được xác định hay không.

Khi một công việc đã được đưa vào một hàng đợi thực hiện, hệ thống sẽ tìm kiếm các bộ xử lí ảo trên các nút tính toán thoả mãn các thuộc tính được yêu cầu bởi công việc đó. Khi các bộ xử lí ảo được tìm thấy, hệ thống sẽ bắt đầu thực hiện công việc trên các nút tìm được.

Các tài nguyên hệ thống của hệ điều hành Linux được hệ thống PBS sử dụng được nêu ra dưới đây:

Tên tài nguyên	Ý nghĩa
Arch	Hệ điều hành của host.n Ví dụ: linux, freebsd, solaris5, aix4
Cput	chỉ ra thời gian cpu theo giây
Idletime	chỉ ra thời gian nghỉ của hệ thống theo giây
Loadave	chỉ ra số các tiến trình đang chạy
Mem	chỉ ra số bộ nhớ đã dùng tính theo byte

Ncpus	chỉ ra số bộ vi xử lý trong máy trạm
Nsessions	chỉ ra số phiên làm việc đang thực hiện trong máy trạm
Nuers	chỉ ra số người dùng có tiến trình đang chạy trong máy trạm
Pids	chỉ ra danh sách các tiến trình của cùng một phiên làm việc.
Physmem	chỉ ra kích thước bộ nhớ vật lý.
Sessions	chỉ ra số phiên làm việc trong máy trạm
Size	chỉ ra kích thước file hệ thống tính theo kb.
Uname	chỉ ra thông tin như lệnh Uname trả về
Validuser	chỉ ra người dùng có hợp pháp không
Walltime	chỉ ra số thời gian tính theo đơn vị giây mà một chương trình hay phiên làm việc tồn tại trong hệ thống

Bảng 7-4 Các tài nguyên hệ thống

Công việc(Job)

Một công việc (Job) là một đối tượng cơ bản nhất được quản lí bởi một hệ thống tính toán theo lô. Theo quan điểm nhìn nhận của người dùng, một công việc là một file kịch bản được chuyển tới hệ thống tính toán theo lệnh mà hệ thống cung cấp. Đây là một file dưới dạng một dạng ngôn ngữ đơn giản, được gọi là shell script được sử dụng rỗng rãi trong hệ điều hành Unix. File này sẽ được biên dịch bởi các lệnh shell. Trên thực tế các file này có thể chứa bất kì dạng dữ liệu gì và người sử dụng có thể yêu cầu bất cứ một chương trình hợp lệ nào xử lý các file này với tư cách là đầu vào chuẩn cho chương trình. Mỗi một công việc sẽ có các thuộc tính mà nó ảnh hưởng tới quá trình xử lý công việc trên Server. Các công việc được duy trì trên Server cho tới khi nó được thực thi hoặc bị yêu cầu chấm dứt. Sau khi nhận file script từ phía người sử dụng, nếu như yêu cầu tạo công việc(Job) tới các đối tượng hàng đợi (Queue) tương ứng theo yêu cầu của người sử dụng là thành công thì hệ thống sẽ tạo ra một công việc(Job) tương ứng. Khi công việc được tạo

ra trên Server, nó được định danh dưới dạng: Sequence_number.Server_name trong đó: Sequence_Number là một số tự nhiên và hai công việc khác nhau, luôn có định danh là khác nhau. Các công việc được tạo ra sau sẽ có Sequence_Number lớn hơn các công việc tạo ra từ trước. Server_Name : là tên Server quản lý công việc đó.

Các thuộc tính của công việc

Một công việc luôn có các thuộc tính công khai được đưa ra trong danh sách sau. Các thuộc tính này có các giá trị mặc định nếu trong trường hợp các thuộc tính của công việc không được đặt các giá trị tương ứng và người sử dụng có thể đặt giá trị cho các thuộc tính này của công việc.

Tên thuộc tính	Định nghĩa
Account_Name	Được sử dụng đối với các tài khoản trên một số máy trạm. Định dạng: String. Giá trị mặc định: rỗng.
Checkpoint	Được cung cấp bởi sự thực thi bởi Server và hệ điều hành của các máy trạm. Thuộc tính này xác định các thời điểm kiểm tra định kì của hệ thống đối với một Job cụ thể nào đó. Giá trị của thuộc tính Checkpoint được sử dụng trong các lệnh người dùng "qalter" và "qsub"(sẽ được đề cập sau).
Error_path	Đường dẫn của file chứa đầu ra chuẩn của lỗi khi thực hiện hệ thống. Định dạng của thuộc tính này có thể được miêu tả dưới dạng: "[Hostname:]pathname" và giá trị mặc định của thuộc tính này là (Job_name).e(Job_number).
Execution_Time	Thời gian chờ đợi để một Job được thực thi. Thời gian này có thể được được tính bằng giây. Nếu thời gian này chưa đạt được, Job sẽ không được lập lịch để thực thi và Job sẽ bị rơi vào tình trạng đợi thực hiện. Định dạng:"M:DD:HH:MM:SS"
Group_list	Một danh sách có dạng group_names@host xác định nhóm mà tại đó một Job cụ thể nào đó được định vị và thực hiện trên một máy trạm được định ra. Khi một Job

	được đưa vào thực hiện, Server sẽ lựa chọn tên của nhóm dựa theo các luật chọn lựa sau đây: 1.Chọn tên nhóm từ danh sách mà sự kết hợp của tên nhóm với tên của máy phù hợp với tên của máy cài đặt thành phần thực thi các tiến trình. 2.Chọn tên nhóm mà nó không đi cùng tên của máy. 3.Sử dụng tên của nhóm là tên đăng nhập của người đăng nhập hệ thống mà dưới tên đăng nhập này chương trình được thực thi. Định dạng của thuộc tính:group_name [@host] [,group_name [@host]...]
Hold_types	Tập hợp các dạng lưu giữ được áp dụng cho các Job. Nếu dạng lưu giữ được đặt là rỗng, Job sẽ không được lập lịch để thực hiện và Job này sẽ rơi vào trạng thái bị lưu giữ.
Job_name	Tên được gắn liền với mỗi một Job được thực hiện thông qua lệnh qsub hoặc lệnh "qalter"(sẽ được đề cập sau). Định dạng chuẩn của tên Job là có độ dài không quá 15 kí tự , kí tự đầu tiên phải là chữ cái. Giá trị mặc định của tên Job dựa vào tên của file kịch bản.
Join_Path	Nếu thuộc tính Join_Path được đặt là true, đầu ra chuẩn của kết quả lỗi khi thực hiện một công việc nào đó sẽ được kết hợp với đầu ra chuẩn của kết quả thực hiện công việc đó và được đặt trong file được xác định bằng thuộc tính Output_Path. Thuộc tính Error_Path được duy trì nhưng lúc này sẽ bị bỏ qua. Định dạng: Boolean và giá trị mặc định là False.
Keep_Files	Nếu thuộc tính Keep_Files được đặt bằng "o" hoặc là "e", kết quả ra cho một Job nào đó sẽ được duy trì trên máy trạm thực hiện Job đó cho đến khi Job đó bị chấm dứt. Giá trị thuộc tính Keep_Files có thể ghi đè lên giá trị của thuộc tính Output_Path và thuộc tính Error_Path. Đinh dạng cho thuộc tính:"o","e","oe","eo" và giá trị

	mặc định bằng rỗng.
Mail_Points	Xác định tại trạng thái thay đổi nào của Job, Server sẽ gửi thông báo về trạng thái hiện thời của Job. Giá trị của thuộc tính này là "a" nghĩa là khi kết thúc hoặc là "b" nghĩa là bắt đầu và giá trị mặc định là "a".
Mail_Users	Tập hợp của các người sử dụng sẽ được gửi mail khi trạng thái nhất định nào đó của Job được đạt tới. Định dạng:user@host[,user@host].
Output_Path	Tên của đường dẫn đầy đủ của file chứa đầu ra chuẩn của kết quả thực hiện một Job nào đó. Định dạng: (Job_name). O.(Job_number)
Priority	Mức độ ưu tiên của Job được thực hiện bởi một người sử dụng nào đó. Định dạng:[+-]priority.
Rerunable	Được đặt là "y " hoặc là "n". Nếu thuộc tính được đặt là "y", Job có thể được chạy lại.
Resource_List	Danh sách các tài nguyên được yêu cầu bởi một Job. Danh sách này là một tập hợp có dạng: tên tài nguyên = giá trị. Nếu các giá trị của các tài nguyên không được đặt, giá trị mặc định được xác định dựa theo các giá trị mặc định của các thuộc tính của đối tượng Queue hoặc đối tượng Server.
Stagein	Danh sách các file đầu vào của các Job.
Stageout	Danh sách các file đầu ra của các Job.
User_List	Danh sách các người sử dụng có dạng , danh sách này xác định tên sẽ được quyền chạy các Job tại các máy tính kết nối ở xa. Khi một Job được đưa vào trạng thái thực hiện, Server sẽ lựa chọn tên người chạy Job dựa vào các luật chọn.
Variable_List	Đây là một danh sách các biến môi trường được đi kèm với Queue hoặc Job.

Ctime	Thời gian mà Job được tạo ra trên Server. Thời gian này được tính theo đơn vị giấy.
Exec_host	Đây là một tập hợp tên các máy mà một Job nào đó đang chạy đồng thời.
Egroup	Nếu một Job nào đó được xếp vào hàng đợi chờ thực hiện, thuộc tính này được đặt bằng tên của của nhóm mà Job được chạy dưới quyền của nhóm đó.
Euser	Nếu một Job nào đó được xếp vào hàng đợi chờ thực hiện, thuộc tính này được đặt bằng tên của của người sử dụng mà Job được chạy dưới quyền của nhóm đó.
Interactive	Nếu thuộc tính này được đặt bằng True, Job có thể tương tác thông qua dòng lệnh.
Job_Owner	Tên truy cập của người thực hiện Job.
Job_state	Trạng thái của Job. Trạng thái náy có thể xác định theo một trong các giá trị sau đây: E: Đã kết thúc. H: Đang bị treo Q: Đang trong hàng đợi thực thi. S: Đang bị dừng. T: Đang trong trạng thái truyền tiếp từ nơi máy này tới máy khác. R: Đang thực hiện. W: Đang trong trạng thái đợi và thời gian thực h[illegible]n với công việc này chưa hết.
Server	Tên của Server quản lí Job.

Bảng 7-5 Các thuộc tính của công việc(job)

7.4.1.3 Tiêu chí thống kê hoạt động của người sử dụng

Về mặt lý thuyết có hai tiêu chí chính để có thể đánh giá và thống kê hoạt động của người sử dụng trong hệ thống. Đó là :

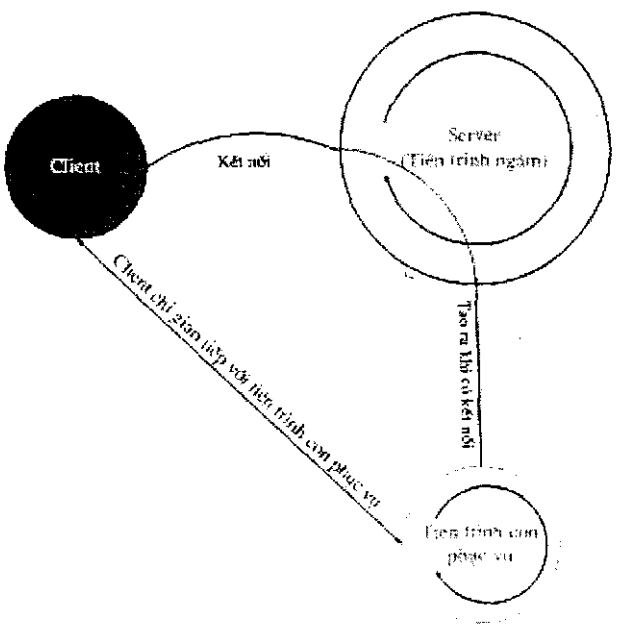
CPU-time : Tổng thời thời gian chiếm giữ CPU của tất cả các chương trình do người dùng đệ trình

Wall-time : Thời gian lâu nhất mà một chương trình có thể tồn tại trong hệ thống

Tuy nhiên chúng ta đều thấy rằng PBS chỉ ghi lại thời gian chiếm CPU tại nút chứa PBS server còn ở các nút tính toán điều này không được ghi lại. Mặt khác, khi chạy các chương trình song song trên các môi trường như LAM-MPI thì các tiến trình do LAM-MPI sinh ra lại vượt ra ngoài tầm kiểm soát của PBS. Do đó dùng CPU-time để đánh giá và thống kê hoạt động của người sử dụng hệ thống là không chính xác và hợp lý, ngược lại với Wall-time thì ở mọi nút tính toán chương trình chỉ tồn tại trong khoảng thời gian cho phép vì thế có thể dùng đại lượng : wall-time * số nút thực thi để đánh gía hoạt động của người sử dụng.

7.5 Xây dựng dịch vụ quản trị hệ thống (server).

7.5.1 Cơ chế sinh tiến trình con phục vụ

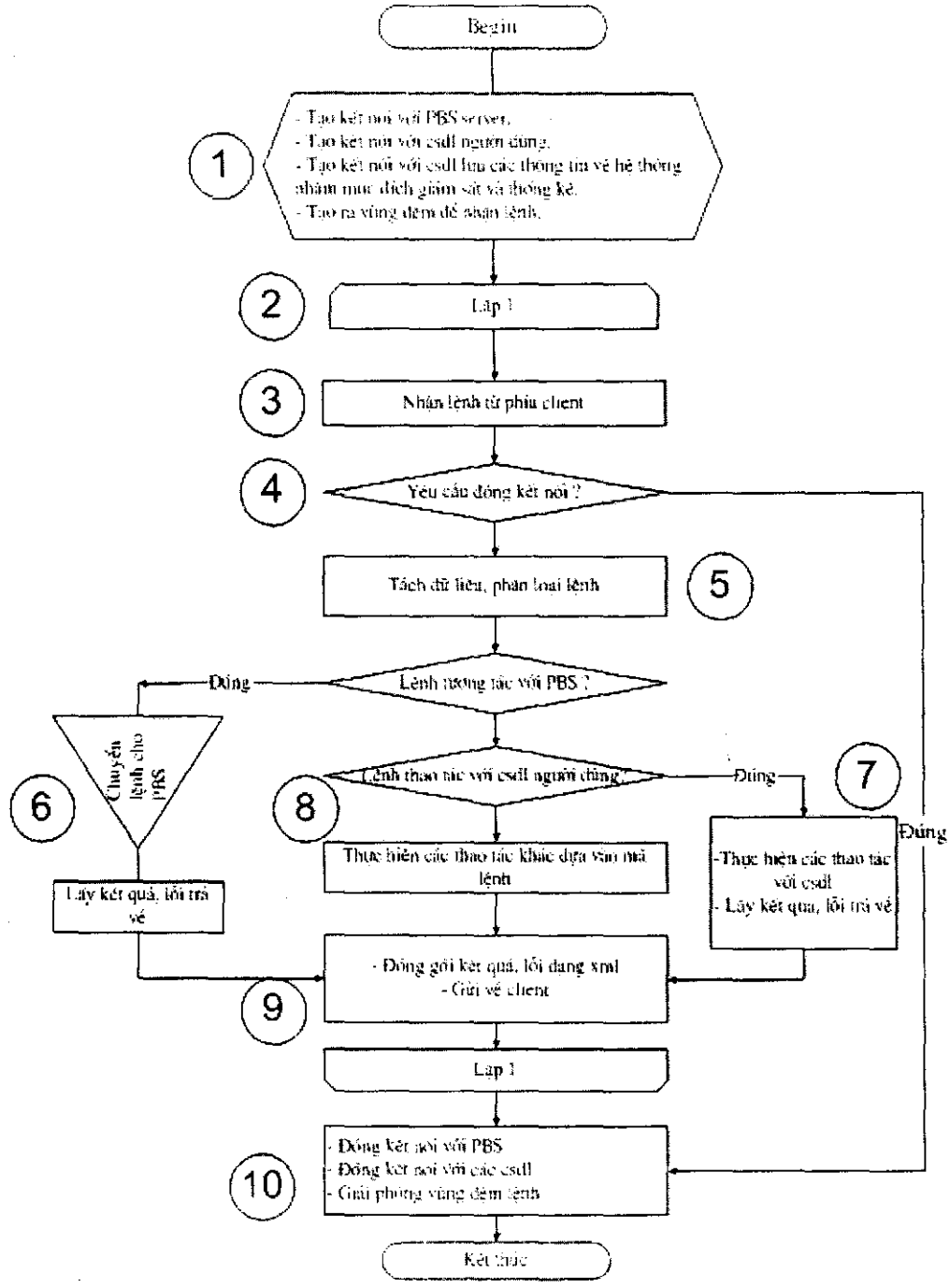


Bảng 7-6 Cơ chế sinh tiến server của server

Theo kiến trúc hệ thống đã xác định, server của bộ công cụ quản trị sẽ được xây dựng là một tiến trình ngầm(deamon) chạy tại máy cài đặt server tính toán (PBS server), vì vậy phương pháp tốt nhất để có thể xây dựng server là hướng cấu trúc tức là server quản trị sẽ được viết bằng ngôn ngữ C trên nền hệ điều hành Linux.

Mỗi khi có kết nối từ phía client thì tiến trình ngầm(server) sẽ sinh ra một tiến trình con phục vụ tương ứng và tiến trình con này sẽ thực hiện đáp ứng các yêu cầu từ phía client. Tới lúc này client sẽ chỉ giao tiếp với tiến trình con phục vụ này, khi kết nối bị đóng thì tiến trình con cũng sẽ tự giải phóng.

7.5.2 Hoạt động của các tiến trình con phục vụ phía trình chủ



Bảng 7-7 Lưu đồ hoạt động của các tiến trình con phục vụ phía trình chủ

Các tiến trình con sẽ trực tiếp phục vụ các yêu cầu từ phía client dưới đây là lưu đồ hoạt động của các tiến trình con phục vụ.

Lưu đồ trên cho thấy chi tiết quá trình tiếp nhận và xử lý các lệnh của một tiến trình con phục vụ

Khi có kết nối tới server, và tiến trình con phục vụ được tạo ra ngay tiếp sau đó tiến trình con sẽ tạo ra các kết nối cần thiết : kết nối với PBS server, kết nối với cơ sở dữ liệu người dùng, và tạo ra vùng đệm để nhận lệnh từ phía client.

Một vòng lặp vô hạn được tạo ra để tiến trình con phục vụ liên tục lắng nghe cổng và đáp ứng các yêu cầu từ phía client. Vòng lặp này sẽ chỉ kết thúc khi có yêu cầu đóng kết nối từ phía client.

Tiến trình con phục vụ lắng nghe và nhận các yêu cầu từ phía client.

- Kiểm tra yêu cầu đó có phải là yêu cầu đóng kết nối hay không, nếu đúng là yêu cầu đóng kết nối thì chuyển tới 10.
- Nếu yêu cầu không phải là yêu cầu đóng kết nối thì thực hiện phân loại lệnh và tách các dữ liệu đi kèm theo.
- Nếu lệnh là lệnh tương tác với PBS thì các cấu trúc dữ liệu tương ứng với thao tác sẽ được thiết lập và chuyển cho PBS thực thi, tiến trình con sẽ chờ cho việc thực hiện kết thúc để lấy kết quả trả về hoặc mã lỗi rồi chuyển đến 9.
- Nếu lệnh là lệnh thao tác trên cơ sở dữ liệu người dùng thì các truy vấn sẽ được thiết lập và thực hiện dựa trên các thông tin cung cấp bởi lệnh
- Nếu lệnh không phải là hai loại trên thì tùy thuộc vào mã lệnh sẽ có xử lý tương ứng. Đây là phần tùy chọn để có thể mở rộng tập lệnh sau này.

Các kết quả và lỗi trả về của các thao tác được đóng gói dạng xml và gửi tới client

Nếu nhận được lệnh yêu cầu đóng kết nối thì tiến trình con phục vụ sẽ đóng các kết nối đã mở và được giải phóng

Lưu đồ trên không thể hiện được một cách chi tiết hoạt động của tiến trình con phục vụ, tuy nhiên đảm bảo được những cấu trúc cơ bản trong tiến trình con phục vụ.

7.5.3 Các thư viện dùng trong server

7.5.3.1 Thư viện libpbs.a

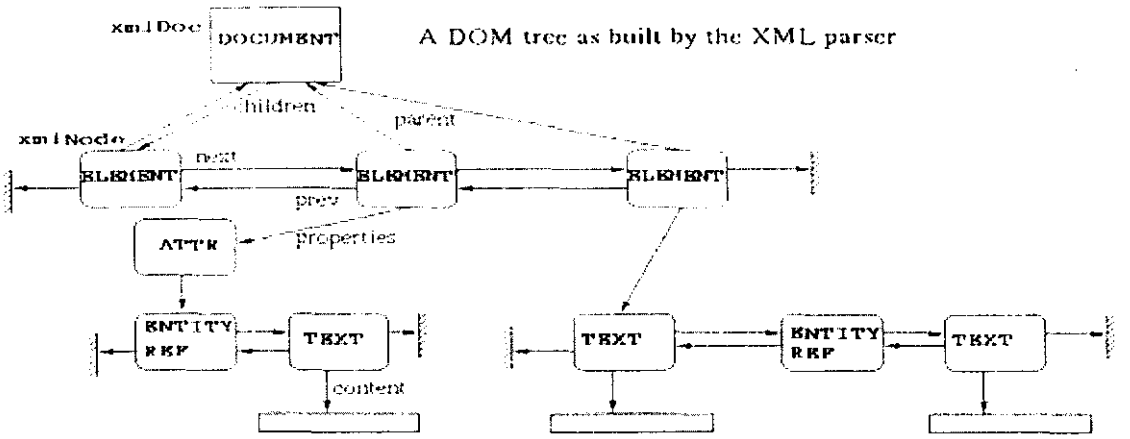
Đây là thư viện cho phép dùng các hàm API(Application Programing Interface) để thực hiện các thao tác với hệ quản lý tài nguyên và phân tải PB. Thư viện này có rất nhiều hàm API, tuy nhiên trong bộ công cụ quản trị hệ thống chỉ sử dụng các hàm API sau :

Tên hàm	Chức năng
Pbs_connect	Tạo một kết nối (socket) tới PBS_Server
Pbs_disconnect	Ngắt kết nối tới PBS_Server
Pbs_geterrmsg	Lấy về thông báo lỗi
Pbs_manager	Thực hiện chức năng quản lý
Pbs_statjob	Trạng thái của job
Pbs_statque	Trạng thái của queue
Pbs_statnode	Lấy về trạng thái của node
Pbs_statserver	Trạng thái của server
Pbs_teminate	Dừng một batch_server

Bảng 7-8 Các hàm sử dụng trong libpbs.a

7.5.3.2 Thư viện libxml2.a

Libxml2 là thư viện được phát triển để sử dụng trong dự án Gnome, tuy nhiên được sử dụng rộng rãi như một thư viện trợ giúp trong việc xử lý các chuỗi với định dạng XML, libxml2 là thư viện rất ổn định và khả chuyển có thể sử dụng trong cả môi trường hệ điều hành Linux, và hệ điều hành Windows. Libxml2 cung cấp cả các phương thức xử lý hướng sự kiện- SAX(Simple API for XML) Interface, và các phương thức xử lý hướng đối tượng- DOM (Document Object Model) Interface, đây là công cụ rất mạnh để xử lý xml trong các ứng dụng hiện nay.



Hình 7-8 Cấu trúc cây khi xử lý xml theo phương thức DOM bằng libxml2

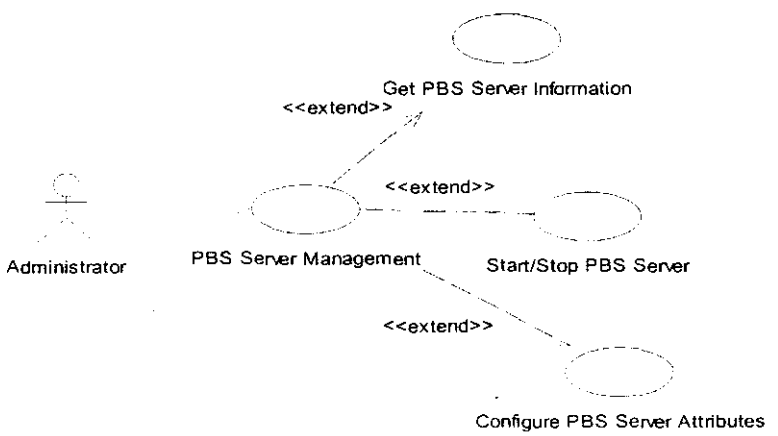
7.6 Xây dựng giao diện quản trị hệ thống(client)

Client được xây dựng với mục đích tạo giao diện với người quản trị và có thể chạy cả trên môi trường Linux và môi trường Windows, sau khi xem xét và đánh giá các giải pháp có thể áp dụng thì việc xây dựng giao diện quản trị hệ thống theo phương pháp hướng đối tượng và sử dụng thư viện Qt của Trolltech là phù hợp hơn cả.

Dưới đây là các use case chính của client.

7.6.1 Các Use case chính

7.6.1.1 Quản lý PBS server



Hình 7-9 Use case quản lý PBS server

Mục đích: Người quản trị cấu hình và quản lý hoạt động của PBS Server.

Tác nhân: Người quản trị, server(tác nhân phụ).

Điều kiện vào:

- Người quản trị đăng nhập vào hệ thống
- Có thể tạo kết nối với server.

Điều kiện ra: Các thay đổi được cập nhật trước khi kết thúc, hoặc nếu có lỗi thì cần thông báo cho người quản trị.

Các trường hợp kết thúc của Use Case

- Kết thúc bình thường mọi thay đổi được cập nhật và hiển thị.
- Có lỗi về mặt thao tác của người sử dụng
- Có lỗi khi thực thi các lệnh ở hệ thống PBS, server trả về lỗi.

Các Use case phụ:

- Lấy thông tin về PBS server(Get PBS Server Information)
- Dừng/kích hoạt PBS server(Start/Stop PBS Server)
- Cập nhật các thuộc tính của PBS (Configure PBS Server Attributes)
- Kịch bản đối thoại của use case phụ (Start/Stop PBS Server)

Stop PBS Server	
	Người quản trị kích vào nút "Stop Server"
	Hệ thống hỏi người quản trị muốn dừng theo cách nào: Dừng tức thì(Stop Immediately) Dừng cho phép kiểm tra(Stop with checkpoint) Hủy bỏ thao tác(Cancel)
	Người quản trị lựa chọn dừng tức thời, chuyển lệnh tới server và đợi cho tới khi có thông báo gửi về từ phía server, hiển thị thông báo cho người quản trị

	Người quản trị chọn dừng với checkpoint vẫn cho phép người quản trị thao tác và chỉ hiển thị thông báo khi có trả lời từ phía server.
	Người quản trị chọn hủy bỏ thao tác(Cancel) hệ thống không có phản hồi cho thao tác này
Start PBS Server	
	Hệ thống đưa ra thông báo, đề nghị người quản trị chờ cho tới khi PBS Server khởi động xong
	Khi có thông báo từ phía server là PBS Server đã khởi động xong thì tắt thông báo và hiển thị thông tin về PBS Server

Bảng 7-9 Kịch bản đối thoại của use case Start/Stop PBS Server

7.6.1.2 Quản lý nút và hàng đợi công việc

Quản lý nút và hàng đợi công việc về cơ bản các thao tác là giống nhau chỉ khác nhau ở loại thuộc tính và số lượng thuộc tính của nút và hàng đợi, dưới đây là use case quản lý nút tính toán

Mục đích: Cấu hình nút và hàng đợi công việc

Tác nhân: Người quản trị, server(tác nhân phụ).

Điều kiện vào: Người dùng phải đăng nhập vào hệ thống và có quyền quản trị hệ thống

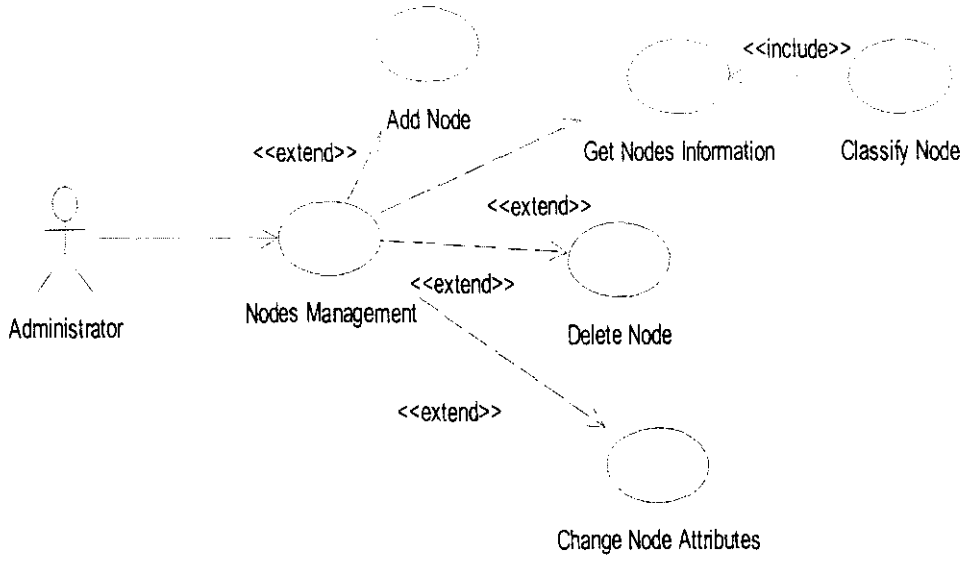
Điều kiện ra: Các thao tác quản lý nút phải được cập nhật và thông tin về trạng thái các nút phải được trình bày.

Các trường hợp kết thúc

- Kết thúc bình thường: các lệnh được thực thi và trạng thái mới của các nút tính toán được cập nhật.
- Người dùng chưa nhập dữ liệu: trước khi thực thi lệnh người dùng chưa nhập dữ liệu
- Nhập dữ liệu sai: Dữ liệu nhập sai, PBS không thực hiện lệnh và trả về lỗi(tên nút không có trong hệ thống, thuộc tính có giá trị sai...)

Các Use case phụ

- Lấy thông tin về nút tính toán(Get Node Information).
- Thêm nút tính toán vào hệ thống(Add Node).
- Xóa nút tính toán ra khỏi hệ thống tính toán bó(Delete Node).
- Cấu hình các thuộc tính của nút tính toán(Change Node Attributes)



Bảng 7-10 Use Case quản lý node tính toán

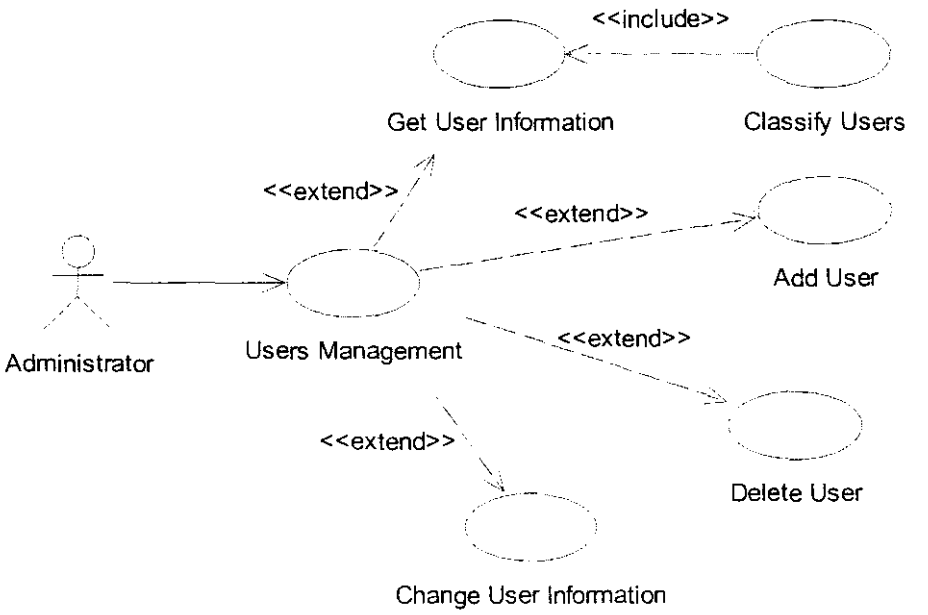
Kịch bản đối thoại của use case: Add Node

Kịch bản thông lệ	
	Người quản trị kích chuột vào nút Add Node
	Hệ thống tạo một một nút mới với tên là: New_Node trên cây phân loại các nút tính toán.
	Người dùng sửa lại tên nút và gõ Enter.
	Hệ thống tạo ra lệnh thêm một nút mới với các thuộc tính mặc định và gửi tới server.

	Thêm một nút mới thành công hệ thống thể hiện thông báo là lệnh đã được thực thi.
Kịch bản mở rộng	
Trường hợp 1:	Người sử dụng nhập tên nút tính toán có dấu ' '.
	Hệ thống thông báo cho người sử dụng: Tên nút không thể chứa ký tự ' '.
Trường hợp 2:	Người sử dụng nhập tên nút tính toán không tồn tại trong hệ thống.
	Server gửi về một thông báo là nút có tên như vậy không tồn tại trong hệ thống.
	Thể hiện thông báo dưới dạng thông báo lỗi.

Bảng 7-11 Kịch bản đối thoại của use case: Add Node

7.6.1.3 Quản lý người dùng



Hình 7-10 Use Case quản lý người dùng

Quản lý người dùng cũng bao gồm các thao tác như là quản lý nút tính toán.

7.6.1.4 Thống kê hoạt động của người sử dụng

Như đã nói ở trên, bộ công cụ quản trị hệ thống sẽ dựa trên tổng số Wall-time của các công việc(job) đối với từng người sử dụng để đánh giá hoạt động của người sử dụng ấy. Mô tả use case thống kê hoạt động của người sử dụng như sau:

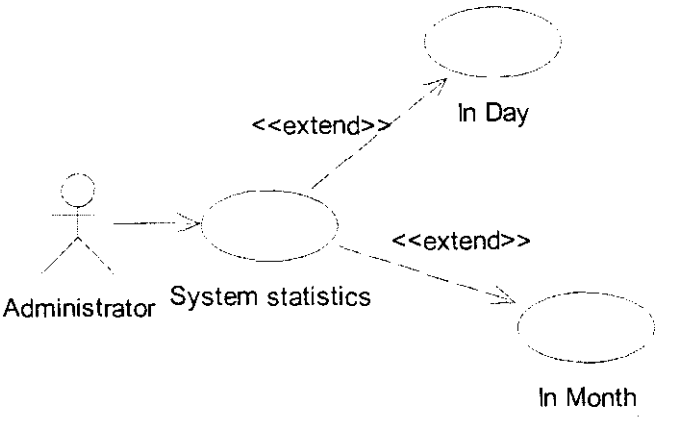
Mục đích: thống kê hoạt động của người sử dụng theo ngày, và tháng

Tác nhân: Người quản trị, server(tác nhân phụ).

Điều kiện vào:

- Người sử dụng cần được xác nhận là có quyền quản trị.
- Tạo được kết nối với server.

Điều kiện ra: không có



Hình 7-11 Use case thống kê hoạt động hệ thống

Các Use case phụ:

- Thống kê hoạt động theo ngày.(In Day)
- Thống kê hoạt động theo tháng.(In Month)
- Kịch bản hoạt động của Use case thống kê theo ngày.

Kịch bản thông lệ	
	Người quản trị chọn ngày, tháng, năm và yêu cầu tạo thống kê
	Hệ thống gửi yêu cầu tới server và lấy lại kết quả.

	Kết quả được thể hiện dưới dạng đồ thị.
Kịch bản mở rộng	
	Không có hoạt động trong ngày được chọn.
	Hệ thống thể hiện thông báo cho người quản trị.

Bảng 7-12 Kịch bản hoạt động của use case thống kê theo ngày.

Về mặt thiết kế cũng như xây dựng hệt thống thì những gì đã trình bày là chưa đủ, tuy nhiên những điểm chính vẫn được đảm bảo, thông tin đầy đủ về việc phân tích và thiết kế và xây dựng sẽ được trình bày ở tài liệu kỹ thuật, ở đây chỉ xin trình bày thêm về giao thức truyền thông sử dụng XML trong việc đóng gói và bóc tách dữ liệu giữa server và client.

7.6.2 Truyền nhận dữ liệu dựa trên XML

XML(Extensible Markup Language) là một tập con của SGML(Standard Generalized Markup Language) được W3C định nghĩa và phát triển, hiện nay XML đang rất được quan tâm như một trong những phương pháp lưu trữ dữ liệu hiệu quả dễ quản lý, và được sử dụng rộng rãi làm giao thức truyền nhận dữ liệu. Tuy việc truyền nhận dữ liệu tuy không phải là quá nhiều nhưng việc sử dụng XML để truyền nhận dữ liệu đã mang lại một sự thuận tiện rất lớn.

7.6.2.1 Định dạng lệnh

Lệnh được đóng trong cặp thẻ <cmd> và </cmd> với các thuộc tính kèm theo như:

- Mã lệnh(code): mã lệnh dùng để qui định cách xử lý ở client khi nhận phản hồi từ phía server.
- Đối tượng tác động(objType): đối tượng tác động là hàng đợi công viêc(queues), nút tính toán(node), hay PBS server.
- Tên đối tượng(objName): định danh của đối tượng tác động.
- Loại lệnh(type): thuộc tính này để xác định cách tác động lên đối tượng tương ứng với các yêu cầu quản lý của PBS: xóa đối tượng, tạo đối tượng, hay thay đổi đối tượng....

Một lệnh gửi tới server không nhất thiết phải có tất cả các thuộc tính trên nhưng

nhất thiết phải có mã lệnh để xác định cách thức xử lý khi nhận phản hồi từ server.

Thông tin về đối tượng tác động được đóng gói trong phần thân của lệnh với các cặp thẻ mang tên các thông tin đó. Sau đây là một ví dụ về định dạng lệnh

<cmd code="6" type="2" objT="0" objN="may22.parallel"> <attr name="resources_available" resource="mem" value="500Mb" /> </cmd>
Lệnh này sẽ được phía server như sau: Mã lệnh: 6 Loại lệnh:2- Thay đổi thuộc tính đối tượng(PBS:Set). Đối tượng tác động: 0- PBS server Tên đối tượng: may22.parallel Thông tin về thuộc tính sẽ được thay đổi: Tên thuộc tính: resources_available Loại: mem- dung lượng nhớ Giá trị thuộc tính: 500Mb

Bảng 7-13 Định dạng Xml của lênh gửi tới server

7.6.2.2 Định dạng phản hồi sau khi thực thi các lệnh

Sau khi thực thi lệnh gửi tới của client, server sẽ gửi trả lời tới client chứa mã lệnh vừa mới thực thi, điều này giúp client có thể hoạt động không đồng bộ với server mà sẽ dựa vào mã lệnh trả về để xử lý.

```
<cmdResponse>
        <header>

<code>2</code>

<resType>0</resType>
```

- Phần header chứa hai thông tin:
- Mã lệnh vừa thực thi
- Loại phản hồi: 0- bình thường, 1- trả về lỗi.

```
</header>
<body>
</body>
</cmdResponse>
```

- Phần body chứa các thông tin tương ứng với các lệnh vừa thực thi . Ví dụ:

```
<cmdResponse>
        <header>
                <code>8</code>
                <resType>0</resType>
                <cmdName>getNodesInfo</cmdName>
        </header>
        <body>
                <deque>
                        <queue_type>Execution</queue_type>
                        <Priority>80</Priority>
                        <total_jobs>0</total_jobs>
                        <state_count>Transit:0   Queued:0   Held:0
Waiting:0 Running:0 Exiting:0 </state_count>
                        <acl_group_enable>True</acl_group_enable>
                </deque>
                <exeque>
                        <queue_type>Execution</queue_type>
                        <Priority>30</Priority>
                        <total_jobs>0</total_jobs>
                        <state_count>Transit:0   Queued:0   Held:0
Waiting:0 Running:0 Exiting:0 </state_count>
                        <route_retry_time>2</route_retry_time>
                </exeque>
                <run_que>
```

```
                        <total_jobs>0</total_jobs>
                        <state_count>Transit:0   Queued:0   Held:0
Waiting:0 Running:0 Exiting:0 </state_count>
                        <acl_group_enable>True</acl_group_enable>
                </run_que>
                <newque>
                        <queue_type>Route</queue_type>
                        <total_jobs>0</total_jobs>
                        <state_count>Transit:0   Queued:0   Held:0
Waiting:0 Running:0 Exiting:0 </state_count>
                </newque>
                <test>
                        <total_jobs>0</total_jobs>
                        <state_count>Transit:0   Queued:0   Held:0
Waiting:0 Running:0 Exiting:0 </state_count>
                </test>
        </body>
</cmdResponse>
```

Bảng 7-14 Sử dụng Xml đóng gói phản hồi lệnh lấy thông tin hàng đợi công việc

resType=0 : không có lỗi	resType=1 : Có lỗi xảy ra
<cmdResponse> <header> <code>34</code> <resType>0</resType> </header> <body> <text>Added User: hpcc</text> </body>	<cmdResponse> <header> <code>34</code> <resType>1</resType> </header> <body> <text>Can't Add User,User existed<text> </body></cmdResponse>

</cmdResponse>	

Bảng 7-15 Sử dụng Xml đóng gói phản hồi lệnh thao tác thêm người dùng

Việc xây dựng các định dạng xml để truyền nhận dữ liệu trong bộ công cụ quản trị tuy mới chỉ tận dụng được những ưu thế cơ bản của xml trong việc đóng gói và bóc tách dữ liệu nhưng đã mang lại những lợi ích sau:

- Dễ xử lý dữ liệu.
- Dễ mở rộng tập lệnh và phản hồi.
- Giúp người phát triển có thể tách biệt quá trình phát triển server và client

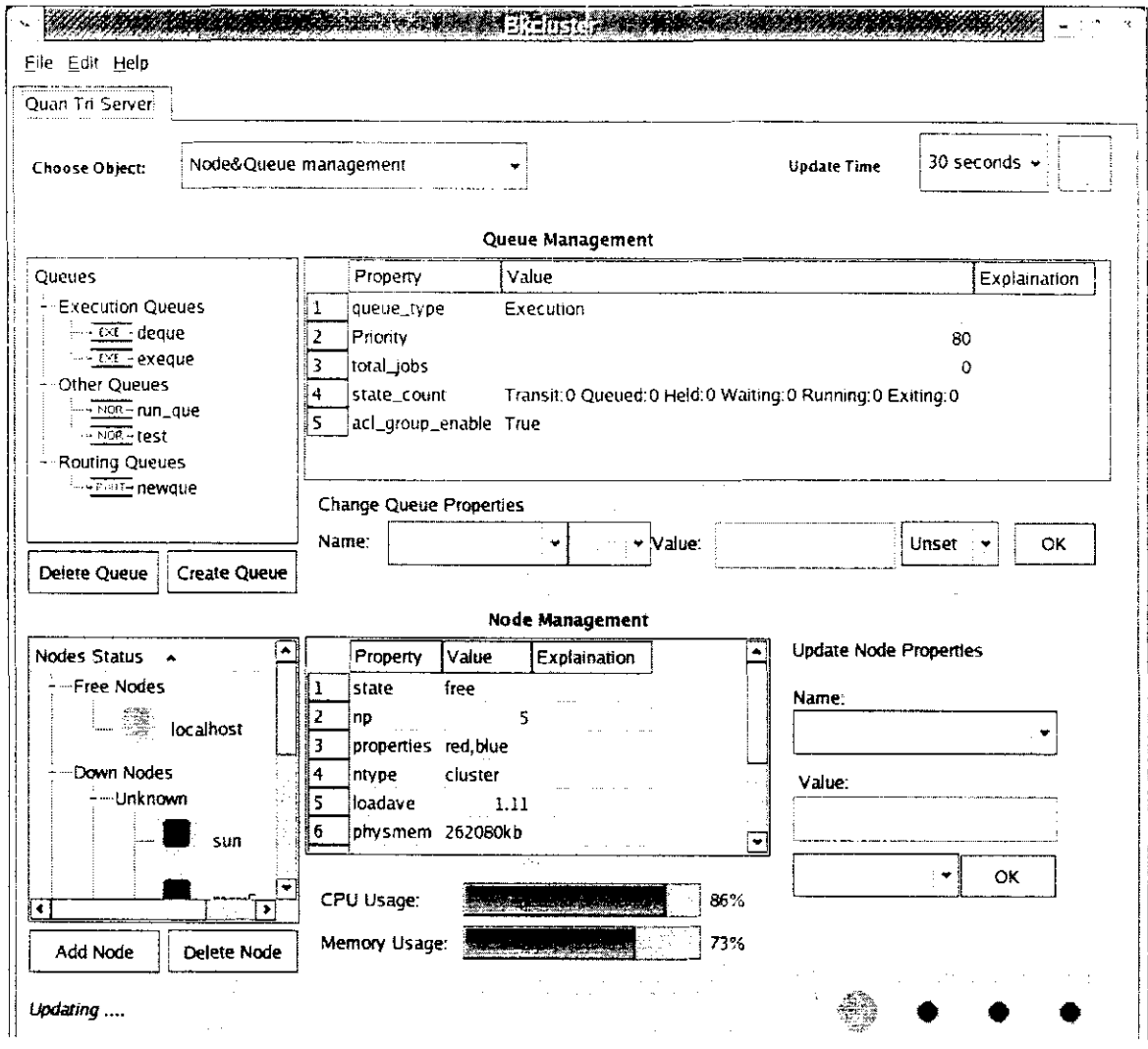
7.7 Kết Quả Đạt Được Và Hướng Phát Triển

Chương cuối cùng này sẽ trình bày về những kết quả đạt được khi tiến hành xây dựng bộ công cụ quản trị hệ thống, đưa ra những hạn chế, những vấn đề chưa giải quyết được, và hướng phát triển trong tương lai.

7.7.1 Các kết quả đạt được

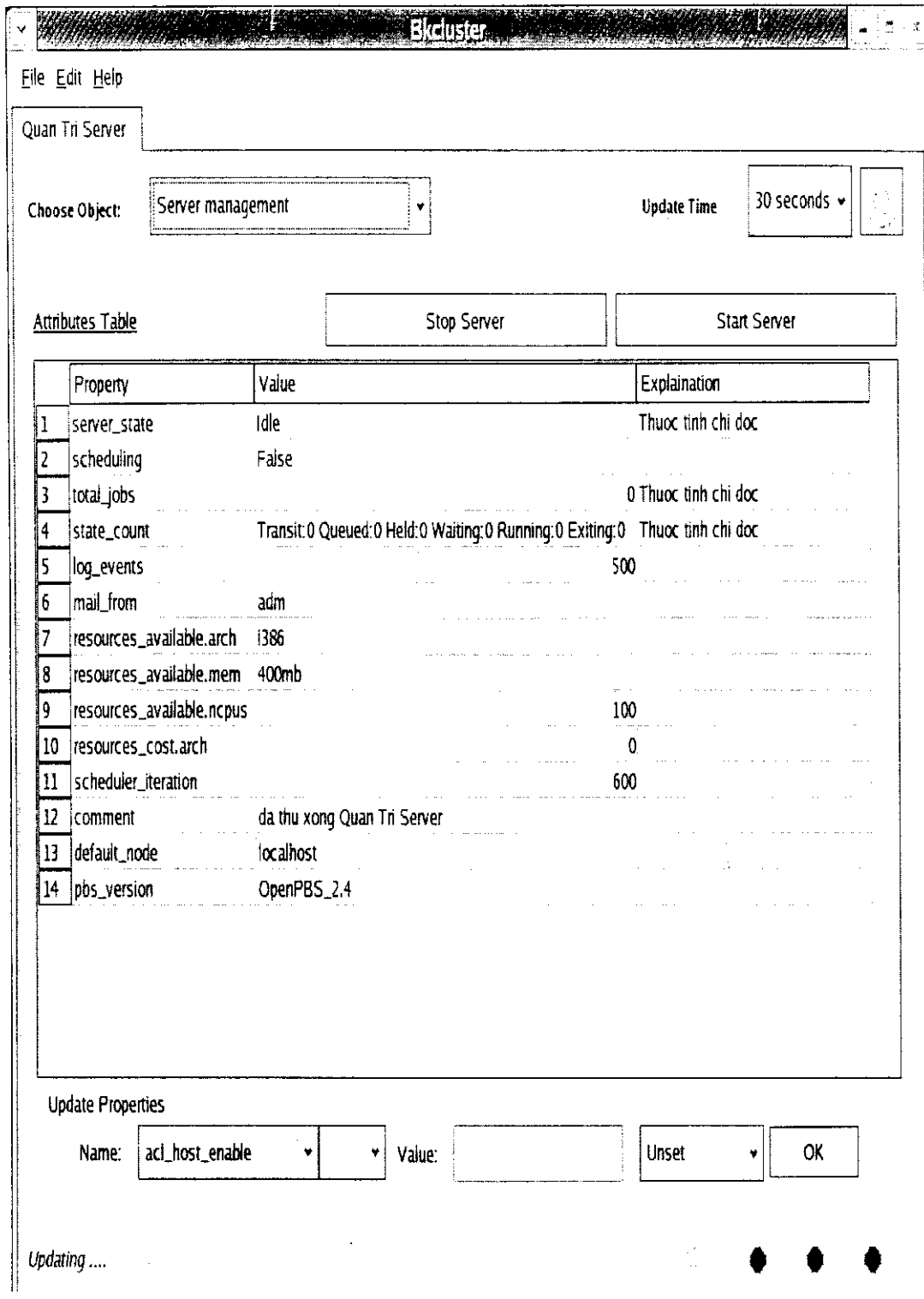
Sau quá trình phân tích, thiết kế và xây dựng, có hai module đã được thực hiện hoàn tất là module dịch vụ quản trị hệ thống viết bằng ngôn ngữ C và module giao diện quan trị hệ thống viết bằng Qt (có thể chạy trên môi trường Linux và Windows):

Giao diện cấu hình và quản lý nút tính toán, hàng đợi công việc



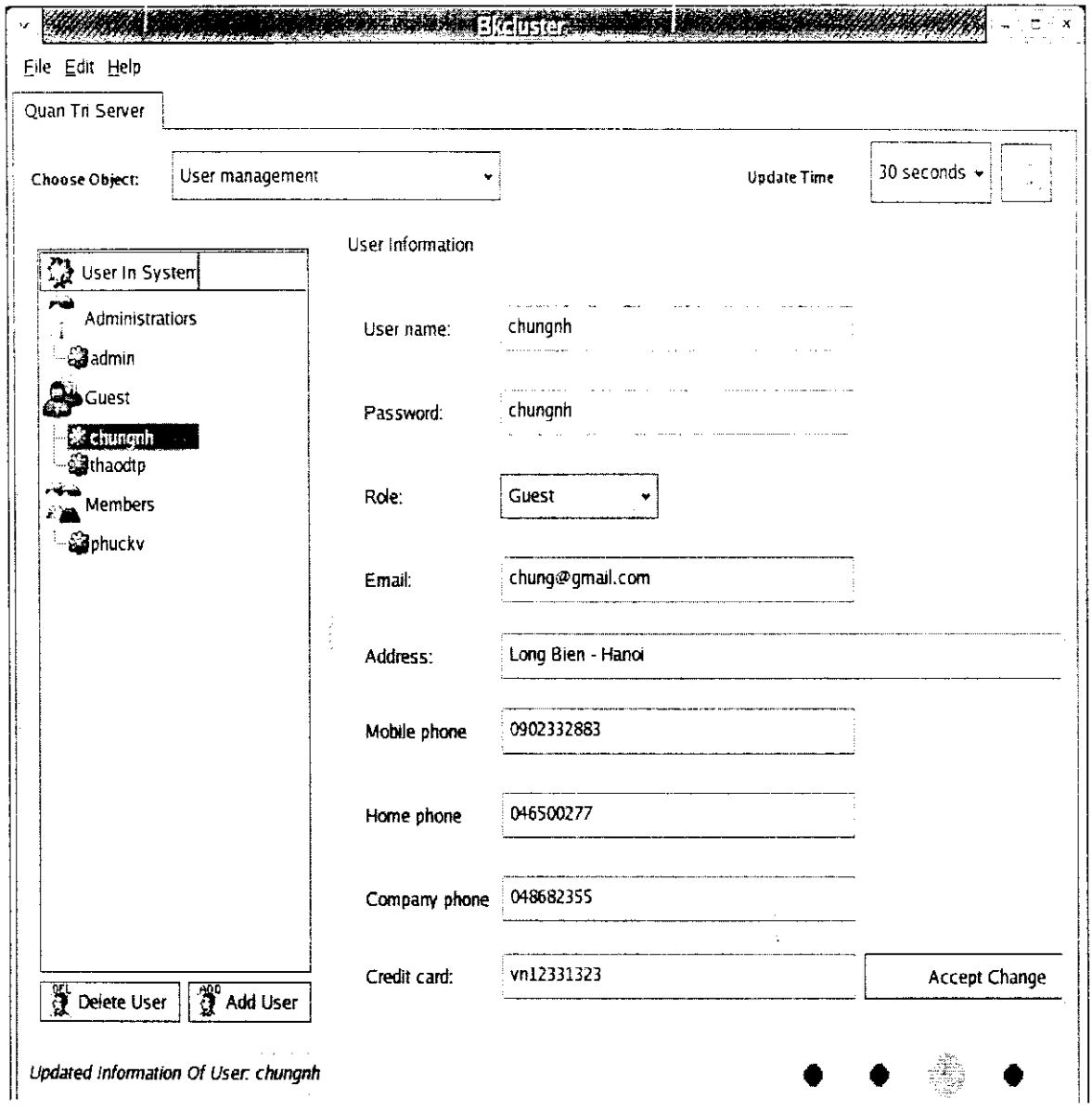
Hình 7-12 Giao diện quản lý node và hàng đợi công việc

Giao diện quản trị server tính toán



Hình 7-13 Giao diện quản lý server tính toán

Giao diện quản lý người dùng



Hình 7-14 Giao diện quản lý người dùng

Về mặt chức năng bộ công cụ quản trị hệ thống đảm bảo được các chức năng cơ bản như đã trình bày trong phần "Phân tích các yêu cầu quản trị". PHÂN TÍCH CÁC YÊU CẦU QUẢN TRỊ

7.7.2 Những hạn chế và hướng phát triển tiếp theo

Bộ công cụ quản trị hệ thống vẫn chưa đáp ứng được các yêu cầu như:

- Sử dụng các chức năng của bộ lập lịch (Maui) (đã cài đặt trong hệ thống cluster tại trung tâm tính toán hiệu năng cao HPCC- ĐHBKHN) để cung cấp cho người quản trị khả năng thống kê và giám sát hoạt động của hệ thống.
- Cung cấp một giao diện có khả năng giải thích ý nghĩa thuộc tính của các đối tượng quản trị.
- Cung cấp một bộ trợ giúp(help).
- Từ những hạn chế nói trên, chúng ta có thể chỉ ra hướng phát triển trong thời gian tới như sau:
- Xây dựng một module cung cấp khả năng thống kê và giám sát hoạt động của người dùng, server tính toán, hàng đợi công việc.
- Cải tiến giao diện để có thể trợ giúp người quản trị về ý nghĩa các thuộc tính của các đối tượng.
- Xây dựng một bộ trợ giúp (help).

CHƯƠNG 8 Module Giám Sát Hoạt Động Hệ Thống

Trong một hệ thống phân cụm, các tài nguyên là các chương trình, các bộ nhớ và các trang bị. Việc quản lý tốt tất cả các loại tài nguyên được coi như một nhân tố chủ yếu để bảo đảm sự vận hành tốt của hệ thống.

Sự quản lý hệ thống có thể chia thành 2 phần chính như sau:

- **Giám sát các tiến trình của hệ thống**: Tất cả cá tiên trình của mỗi nút trong hệ thống phân cụm có thể được theo dõi. Khái niệm "tiến trình" là nền tảng của tất cả các hệ điều hành đa nhiệm như UNIX, WinNT, Linux... Với Unix/Linux, tất cả các job đang thực hiện được thể hiện bởi tiến trình. Một tiến trình là một thực thể bao gồm đồng thời dữ liệu và code. Có thể coi một tiến trình như một đơn vị cơ bản mà nó liên quan các hoạt động trên hệ thống. Để tránh các lỗi và sự tăng phẩm chất của dịch vụ tính toán, tất cả các hệ thống phan cụm phải có một công cụ giám sát việc thực hiện chương trình song song, đặc biệt là các tiến trình của chương trình đó.
- **Quản lý tài nguyên của hệ thống phân cụm** : Ngoài các tiến trình trong cụm, các tài nguyên là cần thiết để quản lý. Các tài nguyên ở đây có thể là tĩnh hay động, phần cứng hay phần mềm. Hệ thống phân cụm là một mạng local các máy tính, cơ chế quản lý chung các tài nguyên của nó là khác nhau với cơ chế quản lý bình thường ở một nút.

Đã tồn tại nhiều công cụ quản lý tài nguyên cho cụm như C3, M3C, SCMS (Smile Cluster Management System), KCAP, Ganglia... nhưng các công cụ mở này có các phần mềm thương mại, lại khó sử dụng. Giao diện người sử dụng của đa số các công cụ này chỉ là màn hình dòng lệnh (console) hay giao diện web. nó yêu cầu nhiều hiểu biết và sự tinh thông của người sử dụng. Vì thế một công cụ mới về quản lý tài nguyên phải trang bị giao diện đồ họa và nội trú trong hệ thông BKluster được quyết định phát triển. Nó là mục tiêu của các công việc trong khuôn khổ của dự án BKluster.

Công cụ quản lý này được phát triển dựa trên Ganglia và RRDTool, sử dụng ngôn ngữ lập trình Qt trên Linux.

- Ganglia là một công cụ quản lý tài nguyên nhưng nó chưa có một giao diện đồ họa. Ganglia có 2 tiến trình ngầm chính: gmond tại mỗi nút tính toán và gmetad ở nút Master.

- RRDTool là một phần mềm để quản lý cơ sở dữ liệu. Nó không chỉ cho phép lưu trữ, quản lý cơ sở dữ liệu mà còn có khả năng tạo các ảnh từ các giá trị trong các file cơ sở dữ liệu.
- QT là một ngôn ngữ lập trình nổi tiếng trong môi trường KDE/GNOME của Linux. Qt là một nền croix-plateforme, một «framework», một người xây dựng giao diện đồ họa (GUI Builder - Graphic User Interface Builder) của sự phát triển các ứng dụng C++ cho việc làm đơn giản trong công việc xây dựng một giao diện đồ họa các ứng dụng.

8.1 Quản lý tài nguyên trong hệ thống phân cụm

Phần thứ nhất của chương này trình bày chi tiết công cụ quản lý tài nguyên chỉ của một máy tính. Các vấn đề đặt ra liên quan đến các tài nguyên của hệ thống phân cụm sẽ được nghiên cứu để hiểu biết hơn các khác nhau giữa công cụ quản lý bình thường và của hệ thống phan cụm. Phần còn lại của chương này quan tâm đến các hàm chính của công cụ quản lý hệ thống này.

8.1.1 Quản lý tài nguyên của một máy tính.

8.1.1.1 Các tài nguyên của một máy tính

Các tài nguyên của một máy tính có thể coi là tất cả mọi thứ trang bị trên máy. Tài nguyên của máy tính có thể được phân thành những loại sau:

- Tài nguyên phần cứng/ phần mềm.
- Tài nguyên tĩnh/ động
- Tài nguyên phần cứng/phần mềm

Theo định nghĩa, "các tài nguyên phần cứng" chỉ định các trang bị máy móc. Các phần cứng được quản lý và theo dõi bởi hệ điều hành, bởi các trình driver. Các tài nguyên phần cứng được chia ra 3 loại:

- Tài nguyên bộ nhớ .
- Tài nguyên processor.
- Tài nguyên đĩa.

Trong khi "tài nguyên phần mềm" tương ứng với hệ điều hành và chúng được cài đặt trên các nút của cụm, chúng cũng được hiểu là các thư viện, các chuẩn và các giao thức, v.v...

Tài nguyên tĩnh/động

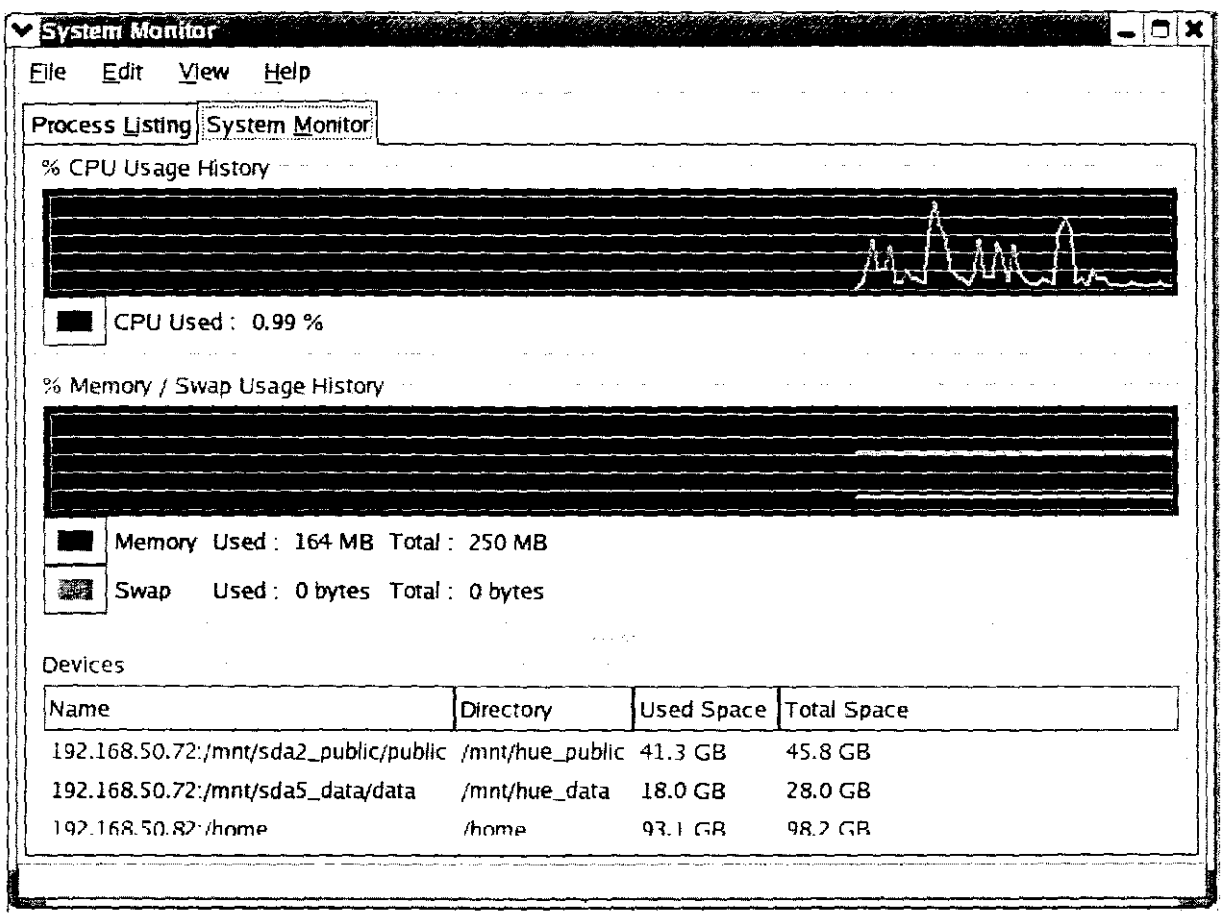
Nói chung, "tài nguyên tĩnh" không thay đổi giá trị trong một quãng thời gian dài như: tốc độ của processor, tổng bộ nhớ, tổng số đĩa, tốc độ truy nhập trong bộ nhớ, tên máy...

"Tài nguyên động" thay đổi với thời gian, ví dụ bộ nhớ bị chiếm đóng, bộ nhớ rảnh rỗi, phần trăm sử dụng processor, số lượng byte gửi , ..

8.1.1.2 Một vài công cụ quản lý tài nguyên trong 1 máy

Các hệ điều hành luôn luôn cung cấp một công cụ quản lý tài nguyên. Công cụ quản lý tài nguyen của 2 hệ điều hành Windows và Linux với một giao diện đồ họa dễ dàng và thuận tiện để sử dụng.

Linux : System Monitor



Hình 8-1: Hệ thống Monitor của Linux RedHat 9

Windows: ***Windows Task Manager***

8.1.2 Tài nguyên của cụm

Một cụm tính toán gồm có một lượng lớn các tài nguyên. Đó là tất cả các tài nguyên tại các nút tính toán và mạng kết nối, chúng hợp thành tài nguyên của cụm. Các tài nguyên này bảo đảm khả năng của cụm: thực hiện tính toán và khả năng lưu trữ. Các tài nguyên của một nút đã nói chi tiết trong phần trước, phần này chỉ tập trung về đặc điểm các tài nguyên của cụm.

Đặc điểm các tài nguyên của cụm:

- Các tài nguyên rất lớn và khác biệt.
- Các tài nguyên (vật lý và phần mềm) được phân bố trên các nút khác nhau của cụm.
- Các tài nguyên của một nút chỉ có thể được sử dụng nếu nó chưa được sử dụng trong một công việc nào

Một nút có thể không tránh khỏi các lỗi. Nhưng lỗi của một nút phải không ảnh hưởng đến các nút khác. Nghĩa là trong trường hợp một hay vài nút phải khởi động lại chẳng hạn, cụm vẫn hoạt động tốt.

8.1.3 Sự hoạt động của một công cụ quản lý tài nguyên cụm

Phần này tổng kết các chức năng chính và các nguyên tắc chung của quản lý tài nguyên của hệ thống phân cụm. Vấn đề đặt ra của quản lý tài nguyên của cụm là không đơn giản và dễ dàng. Một công cụ quản lý tài nguyên của hệ thống phân cụm có thể quản lý cụm như một thực thể duy nhất, nghĩa là nó phải có các khả năng sau :

- Quản lý tài nguyên của mỗi nút trong hệ thống phân cụm.
- Quản lý mạng kết nối
- Đánh giá hiệu năng của hệ thống.

8.1.3.1 Quản lý các tài nguyên của mỗi nút của hệ thống phân cụm.

Hiệu quả thực hiện chương trình song song tùy thuộc nhiều yếu tố: các nhân tố khách quan và chủ quan. Các nhân tố chủ quan là các thuật toán mà người ta chọn và sử dụng, cách chạy chương trình .. Cách ở đây là chọn các nút để sử dụng - là sự quyết định của người sử dụng. Nếu các nút tự do (ví dụ: phần trăm sử dụng của nút < 25%) được chọn, thời gian thực hiện sẽ nhanh hơn trường hợp các nút đã bị

quá tải (ví dụ: sử dụng > 100%) được chọn. Phần khác, chương trình song song đang thực hiện có thể trả lại vài vấn đề lỗi. Đó là các thông tin có ích về trạng thái hiện tại của chương trình đang chạy.

Công cụ xây dựng có khả năng quản lý cấu hình mỗi nút tính toán: cấu hình phần cứng và phần mềm. Công việc quan trọng nhất của công cụ này là quản lý các thông tin liên quan đến tài nguyên của mỗi nút tinh toán. Các thông tin tài nguyên này của phần cứng hay phần mềm có thể là tĩnh hay động, như đã nói trong phần trước. Vì với các thông tin chi tiết lien quan mỗi nút, người sử dụng có thể tưởng tượng cái nhìn tổng quát của cụm mỗi khi cập nhật hệ thống. Các thông tin giám sát là :

- Số lượng tổng nút, số lượng các nút hoạt động, không hoạt động.
- Trạng thái hoạt động hay không của mỗi nút.
- Tài nguyên bộ nhớ của một nút và của cụm.
- Tài nguyên processor của một nút và của cụm.
- Tài nguyên đĩa của một nút và của cụm.
- Yêu cầu hệ thống của một nút và của cụm.

Thêm vào dó, sự quản lý tài nguyên của cụm có thể thực hiện từ xa. Như mọi người đã biết, hệ điều hành cung cấp khả năng truy cập từ xa các tài nguyên của một máy tính. Linux và Windows đưa ra một tùy chọn đa dạng. Hầu hết các phiên bản Linux cung cấp SSH, telnet và ftp. Sự tích hợp bảo mật của Web của Linux có thể là hoàn thiện, với sự giúp đỡ ví dụ của server Web Apache. Nhưng các cơ chế này không đơn giản, dễ dàng để sử dụng, với một cụm nhiều nút. Ví dụ, sự quản lý tài nguyên của một cụm có 1000 trạm là một công việc nặng nhọc.

8.1.3.2 Quản lý mạng kết nối

Mạng kết nối là một phần cần thiết của cụm. Nó chắc chắn là một mạng tốc độ cao. Sự quản lý trạng thái của mạng cũng là công việc quan trọng để đảm bảo các hoạt động của hệ thống. Trạng thái mạng bao gồm nhiều tham số, như tốc độ, lưu lượng, thời gian phản ứng .. Ở đây, công cụ quản lý tài nguyên gắn bó quan trọng với giám sát lưu lượng mạng, số lượng byte gửi và nhận.

Theo các thông tin này trên trạng thái của mạng cung cấp bởi công cụ quản lý tài nguyên, các kết nối trong hệ thống có thể được giám sát. Nó có ích làm giảm bớt sự nguy hiểm và sửa chữa các lỗi, đặc biệt trong thời gian chạy chương trình, vì

chương trình song song cần rất nhiều sự truyền thông.

8.1.3.3 Đánh giá hiệu năng của hệ thống phân cụm

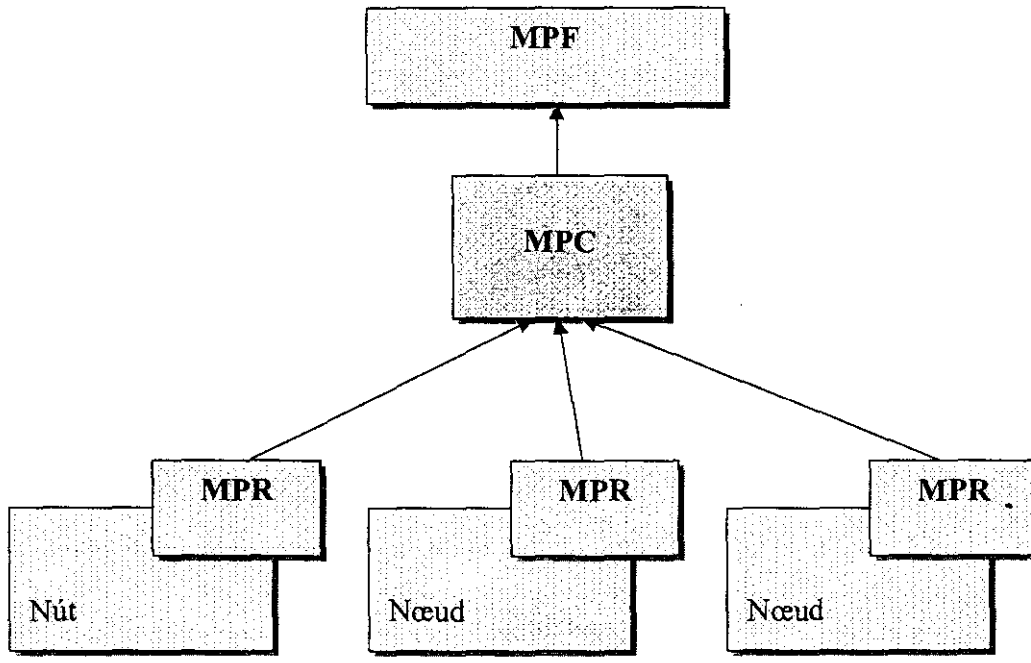
Lĩnh vực này đã được nghiên cứu tỉ mỉ trên thế giới. Có rất nhiều công cụ để đo hiệu năng của hệ thống, ví dụ NPB (NAS Parallel Benchmark). Khả năng của hệ thống phân cụm có thể được đánh giá khách quan.

Đối tượng đo lường hiệu năng là processor, mạng, hệ thống file, vào/ra song song hay có thể là thư viện MPI, hay hiệu năng của công cụ biên dịch của ngôn ngữ C, FORTRAN...

8.1.4 Khó khăn - giải pháp

Đó là hiểu biết có ích trạng thái của cụm bất kì lúc nào, tại thời điểm chạy chương trình. Nhưng người ta có thể quản lý cụm như thế nào trong khi hệ điều hành không cung cấp công cụ để theo dõi tự động tất cả các nút tính toán.

8.1.4.1 Cấu trúc công cụ quản lý tài nguyên của cụm



Hình 8-2 Hai module quản lý cụm

Sau khi có hiểu biết kĩ về đặc điểm các tài nguyên của cụm, người ta đưa ra một giải pháp để quản lý tài nguyên của cụm như một thực thể duy nhất. Phải có 3

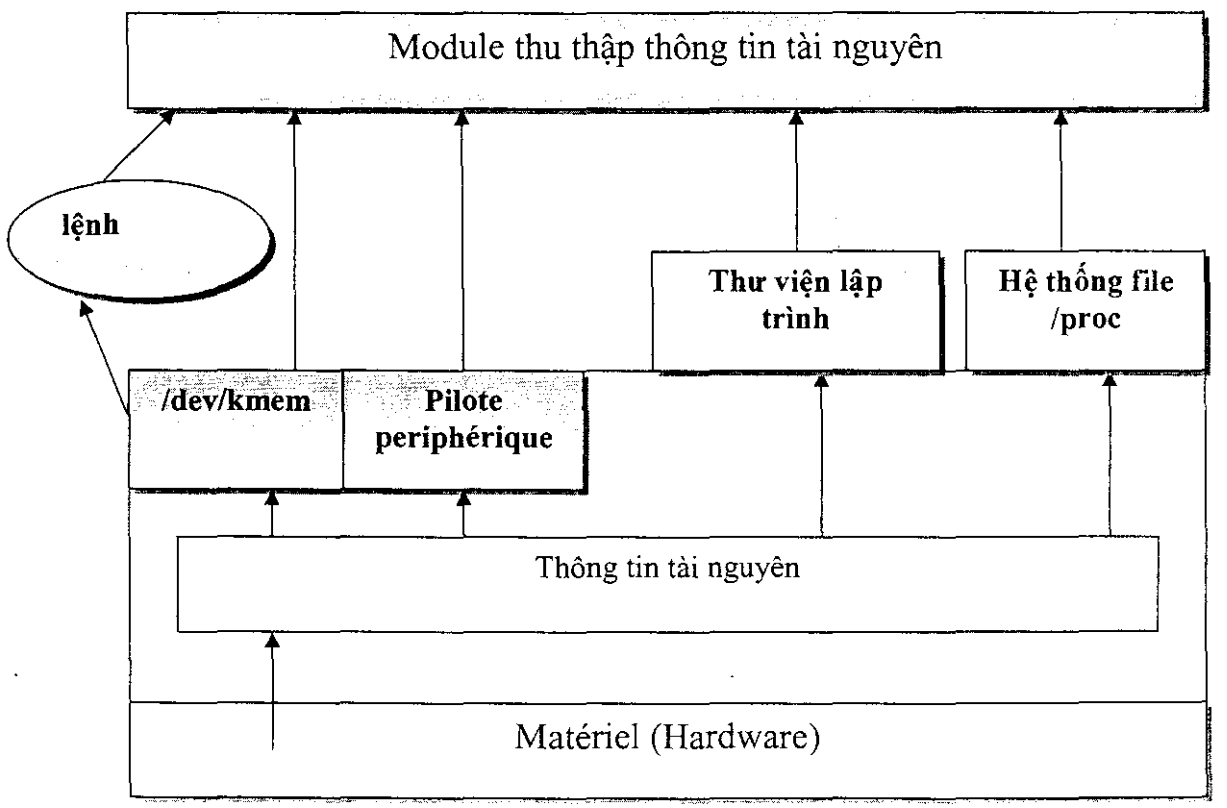
module:

- Module MPR để tập hợp các thông tin của tài nguyên ở tất cả các nút tính toán với hệ điều hành địa phương.
- Module MPC ở một nút Master để sưu tập các thông tin các module MPR và lưu.
- Module MPF để hiển thị các thông tin được sưu tập bởi module MPC, cho dưới dạng text, dạng đồ họa.

8.1.4.2 Phương thức thu thập thông tin tài nguyên trong Linux

Hạt nhân của hệ điều hành (Linux hay Windows) quản lý trạng thái của máy tính và đảm bảo tất cả các hoạt động trong máy tính này. Nội dung phần này sẽ trình bày về các phương thức truy nhập thông tin các tài nguyên của một máy dùng Linux.

Có 4 phương thức được minh họa các phương thức để thu thập thông tin tài nguyên trong một máy.



Hình 8-3 Phương thức thu thập thông tin của một nút

- **Truy nhập trực tiếp bộ nhớ của hạt nhân**

Trong cách này, các dữ liệu đạt được bởi một địa chỉ đọc trực tiếp đã biết (sử dụng một giao diện như /dev/kmem). Cách này thu thập các thông tin về tùy thuộc phiên bản của hạt nhân. Một sự thay đổi nhỏ của hạt nhận có thể gây lỗi cho processor thu thập các thông tin tài nguyên. Để tránh vấn đề này, phần lớn các hạt nhân hiện đại cung cấp thường xuyên một giao diện thông tin của hệ thống API. Ưu điểm của cách này là tốc độ của thu thập thông tin.

Vài công cụ mới đây được phát triển biết truy nhập nhanh thông tin hạt nhân sử dụng một ccông cụ quản lý bên ngoài client (custom driver) mà nó vào trực tiếp tới tất cả thông tin bằng giao diện ứng dụng bảo vệ API cố định. Ưu điểm cua cách này là truy nhập nhanh và độc lập với hạt nhân.

- **Hệ thống file /proc :**

Phần lớn các hệ thống UNIX, nhu Linux, có một giao diện ảo để làm việc thu thập các thông tin tài nguyên dễ dàng hơn, đó là hệ thống file /proc. Các thông tin bên trong hạt nhân như đơn vị trung tâm, các tiến trình, các file, các trạng thái mạng .. đặt trong /proc. Nó có khả năng để nhìn bao quát thư mục /proc để tìm kiếm các thông tin mong muốn. Cách này cho phép di chuyển tốt hơn và một nhân độc lập. Người ta cũng chú ý tốc độ truy nhập có thể so sánh truy nhập trực tiếp trong hạt nhân vì đó là coi như truy nhập trực tiếp trong bộ nhớ của hạt nhân . Dự án phát triển một lớp các thư viện trên hệ thống file /proc nó cho phép một sự di truyển tốt hơn và dễ dàng lập trình.

- **Thư viện lập trình của Linux :**

Hệ điwuf hành Linux cung cấp các thư viện để thu thập thông tin của một máy. Ưu điểm của phương pháp này là dễ dàng lập trình và một hạt nhân độc lập. Trong lập trình với các thư viện này, chúng ta có các hàm mà chúng có thể gọi dễ dàng gần như với mọi ngôn ngữ lập trình và script dựa trên C hay C++.

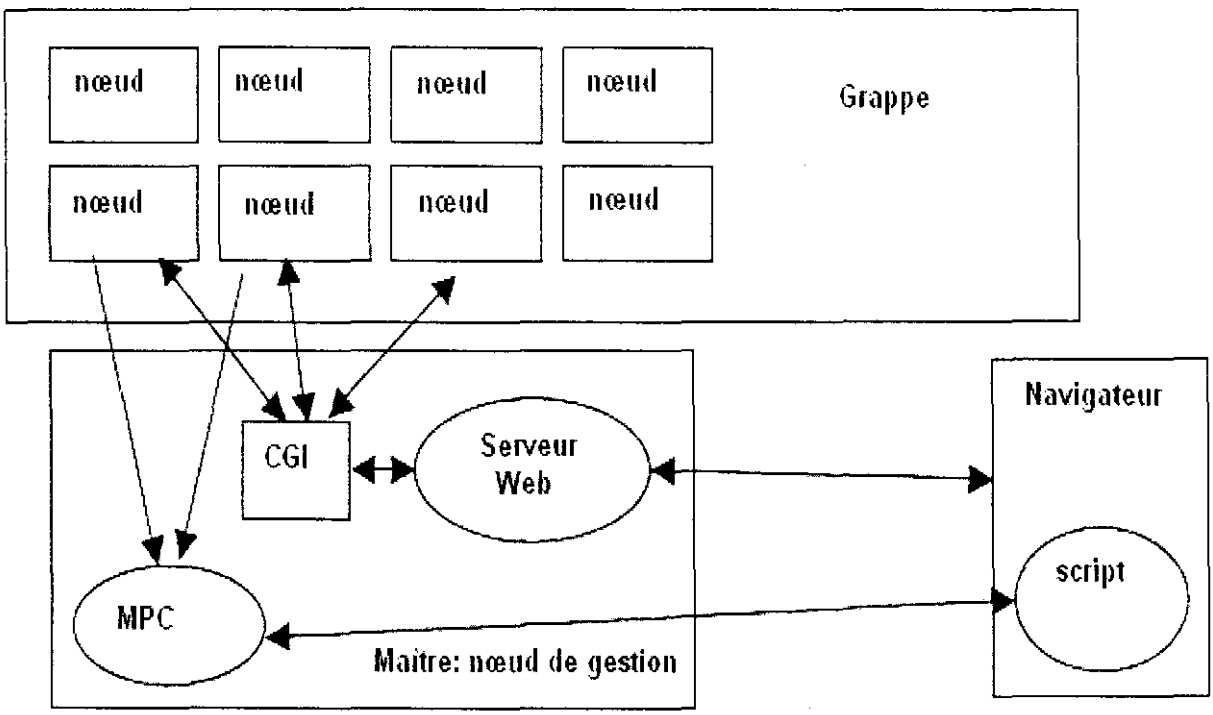
- **Lệnh của hệ thống :**

Vài công cụ quản lý dựa trên UNIX/Linux sử dụng kết quả chạy các lệnh chuẩn của UNIX/Linux như ps hay uptime... Nó cung cấp khả năng chuyển đổi giữa các họ các hệ thống Linux. Đó là một cách không hiệu quả vì yêu cầu cao trên hệ thống mỗi lần mà các lệnh được chạy.

8.1.4.3 Công nghệ web để làm nổi rõ các thông tin tài nguyên của cụm

Module MPC thu thập thông tin tài nguyên của tất cả các nút tính toán của cụm sau đó, module MPF phụ trách hiển thị các thông tin này.

Công nghệ thường được sử dụng nhiều là web. Người sử dụng và người quản trị quản lý cụm qua một trình duyệt.



Hình 8-4 Cấu trúc hệ thống quản lý và theo dõi cụm dựa trên web

Phần lớn các công cụ quản lý tài nguyên trên thực tế là một giao diện web với web server phổ biến là Apache.

8.1.5 Ví dụ một công cụ quản lý tài nguyên của cụm

Ta đã nghiêm cứu tập các công cụ quản lý tài nguyên của cụm thông dụng trên thế giới như C3, M3C, SCMS, KCAP, Ganglia...

8.1.5.1 Ganglia

Ganglia là một dự án mở (open-source project) nó được phát triển ở trường đại học California, Berkeley. Nó là công cụ giám sát hệ thống phân bố như cụm hay lưới tính toán. Hệ thống Ganglia có 2 tiến trình ngầm chính (gmond và gmetad),

một giao diện web dựa trên (PHP-based Web frontend) và một vài chương trình nhỏ khác. 3 tiến trình ngầm chính như các module MPR (Module để thu thập các thông tin tài nguyên của một nút của mmột hệ thống), MPC (Module để tích cóp các dư liệu của MPR) và MPF (Module để hiển thị các thông tin tài nguyên) trong cấu trúc của công cụ quản lý tài nguyên.

Gmond (Ganglia Monitoring Daemon)

Để giám sát các tài nguyên của hệ thống phân phối như lưới hay đặc biệt là cụm tính toán, phải theo dõi tốt các tài nguyên của tất cả các máy tính trong hệ thống như : các thông tin như số lượng các nút trong hệ thống phân cụm, trạng thái của mỗi nút trong thời điểm hoat động, tên của nút, tốc độc CPU, số lượng processor của mỗi nút, bộ nhớ tự do, tổng bộ nhớ, số lượng byte nhận/ gửi, ...

Gmond (Ganglia Monitor Daemon) là một trong 2 tiến trình ngầm chính của ganglia, nó chạy qua trên mỗi nút của hệ thống mà chúng ta muốn theo dõi.

Gmond có 4 trách nhiệm chính:

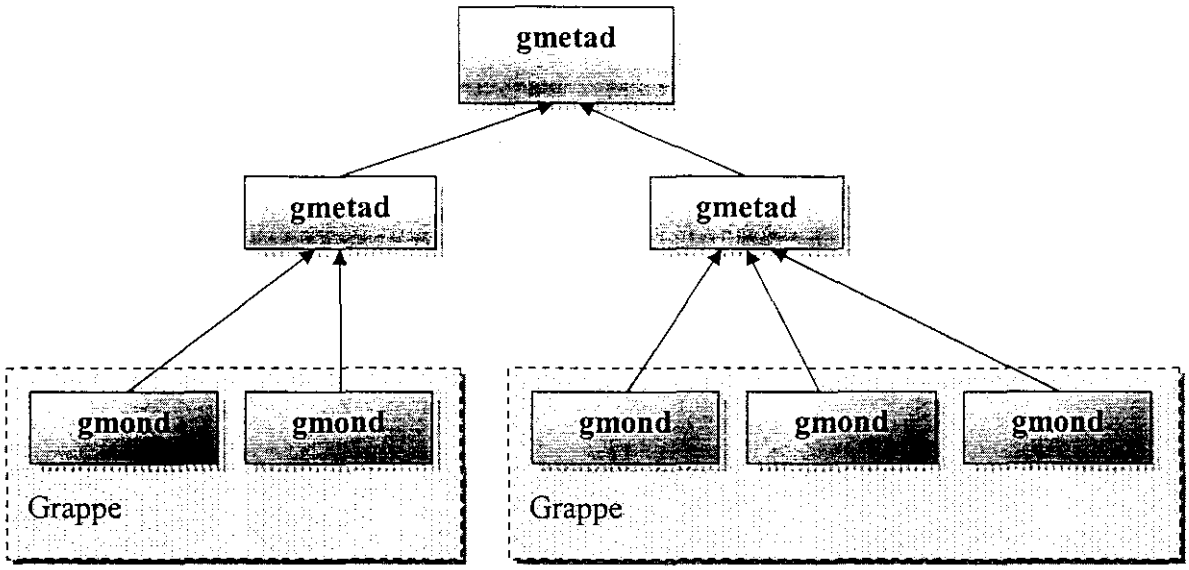
- Quản lý sự thay đổi trong trạng thái của một nút.
- Báo hiệu các thay đổi thích hợp
- Nghe trạng thái của tất cả các nút khác của cụm.
- Đáp lại các yêu cầu cho một mô tả XML của trạng thái của cụm.

Để thu thập các thông tin tài nguyên một nút của cụm, gmond sử dụng giao diện hệ thống file /proc của Linux. Mỗi gmond truyền các dữ liệu với 2 cách khác nhau: trong định dạng ngoài của dữ liệu (XDR) sử dụng các message UDP hay gửi XML bởi một kết nối TCP.

Gmetad (Ganglia Meta Daemon)

Tiến trình ngầm này phụ trách thu thập các dữ liệu của tất cả các gmond. Sự kết hợp giữa 2 thành phần gmond và gmetad được thực hiện bởi việc dùng một cây kết nối điểm - điểm giữa các nút tiêu biểu của cụm . Từ gmetad, có thể thấy và quản lý trang thái của cụm. Tại mỗi nút trong cây, một tiến trình ngầm gmetad nhận định kì một bộ các nguồn dữ liệu của nút con (gmond hay có thể là gmetad), phân tích XML thu được, bảo toàn tất cả các luật trong cơ sở dữ liệu Round Robin và xuất ra XML đến các module client bởi 1 socket TCP. Gmetad cho phép quản lý một cụm và các tập các cụm. Các nguồn dữ liệu có thể là các tiến trình ngầm gmond hiển thị cụm chuyên biệt, hay các gmetad khác hiển thị các tập các cụm.

Gmetad sử dụng các địa chỉ của nguồn IP để điều khiển truy nhập và mỗi IP đặc thù cho một cụm.



Hình 8-5 Cấu trúc của Ganglia

Cổng gmetad sử dụng để xuất các dữ liệu XML là 8652 (mặc định).

PHP-based Web Frontend

Một giao diện web là chỗ mà các thông tin tài nguyên của cụm được hiển thị. Các thông tin được hiển thị có 2 loại: text và đồ họa. Người sử dụng có thể thấy các tài nguyên của cụm hay một nút của hệ thống trong một giờ sau/ một tuần sau/ một ngày sau/ một tháng sau/ hay một năm sau, bởi một trình duyệt (browser) như Internet Explorer, Firefox, Mozilla...

8.1.5.2 C3 & M3C

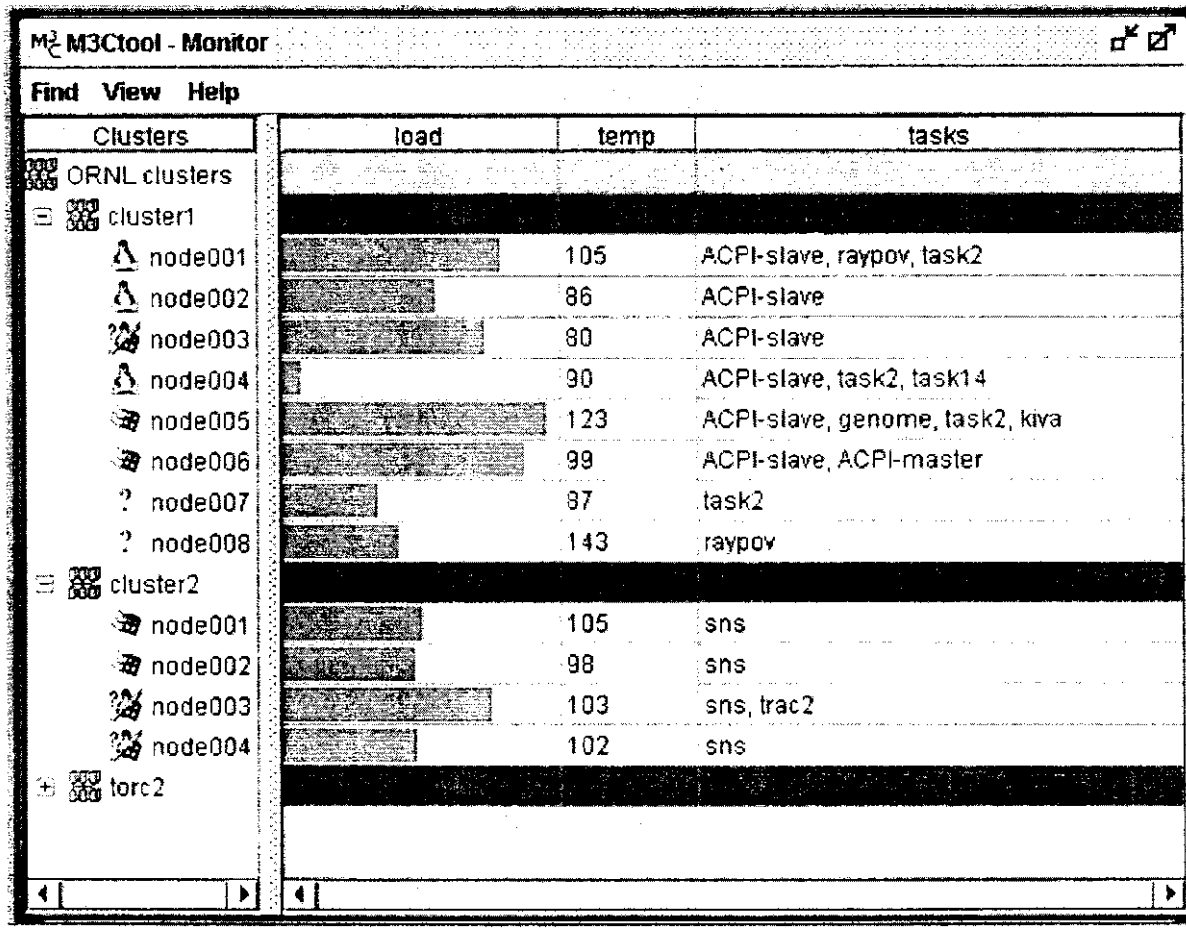
C3 (Cluster Commande Control) và M3C (Monitoring and Managing Multiple Cluster) là 2 công cụ thường được sử dụng hiện nay, ví dụ trong hệ thống « HighTORC Linux Cluster » của phòng thí nghiệm quốc gia Oak Ridge.

C3 là một công cụ quản lý và theo dõi các cụm tính toán với một giao diện dòng lệnh. Có 8 lệnh cơ bản:

- cl_pushimage
- cl_shutdown

- cl_push
- cl_rm
- cl_get
- cl_ps
- cl_kill
- cl_exec

M3C có 6 công cụ tích hợp. Nó cung cấp một giao diện đồ họa web cho người sử dụng và người quản trị cụm. a six outils d'intégration.



Hình 8-6 Giao diện web của M3C

8.1.5.3 SCMS

SCMS (Scalable Cluster Management System) là một công cụ quản lý tương tác và co giãn của cụm. Mục đích của SCMS là cho phép người sử dụng thực hiện dễ dàng các công việc quản trị. SCMS cung cấp một số lượng lớn (commande tool),

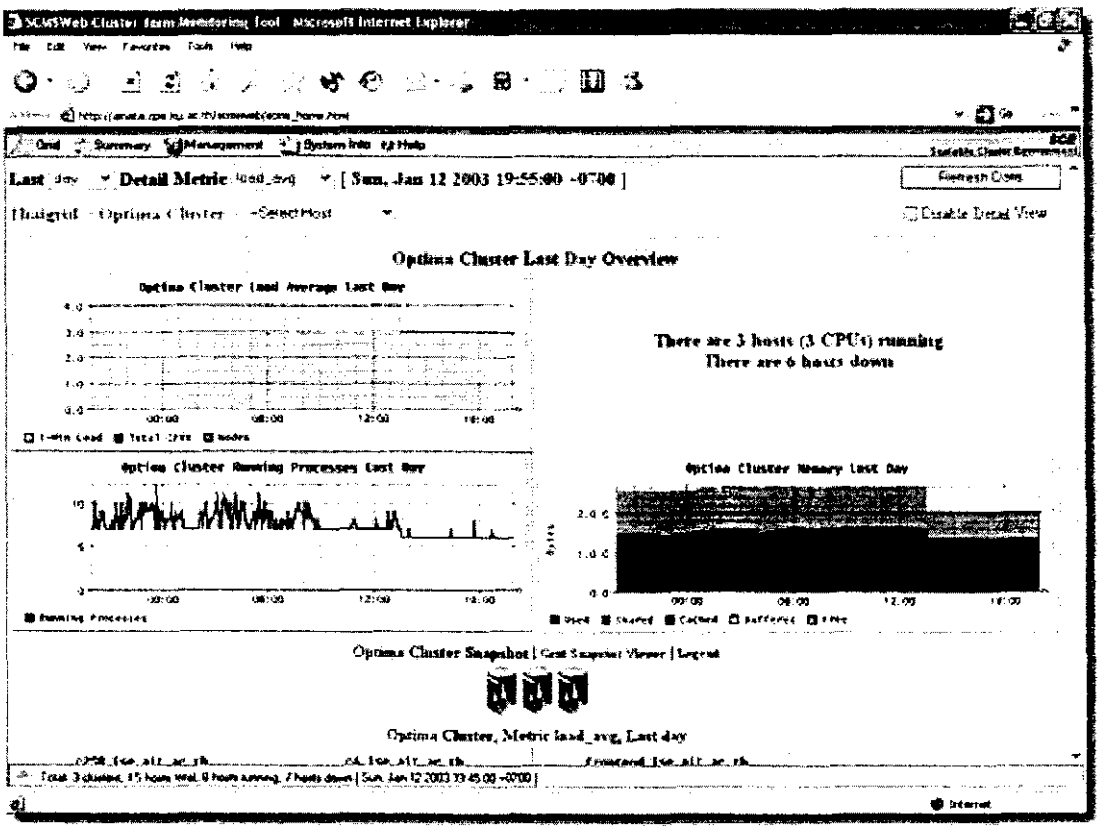
hệ thống con của việc giám sát thời gian thực, một giao diện web .. Với SCMS, các công việc quản trị hệ thống phân cụm trở nên đơn giản hơn rất nhiều.

Công cụ lệnh

- scms_initconf là một công cụ để cấu hình dễ dàng cho SCMS. Nhà quản trị ít ra cũng phải biết danh sách các tên host.
- sce_host giúp đưa ra các nút trên dòng hay tất cả các nút tính toán.
- sce_rms : giao diện của sce_rms để các hệ thống con của giám sát thời gian thực. Nó cung cấp các chức năng của tất cả việc giám sát.

Giao diện Web

Có 3 cấp nhìn : cấp lưới, cấp cụm và cấp nút.



Hình 8-7 Giao diện web quản lý cluster cấp độ hệ thống

8.1.5.4 KCAP

KCAP là một công cụ dựa trên công nghệ web. Nó có một tập các công cụ và các script cho phép chuyển đổi cấu hình phức tạp của hệ thống hiển thị 3D. Nó cho

phép người sử dụng dùng có hiệu quả hơn để quản lý và làm thấy rõ hệ thống phân cụm. KCAP sử dụng với chương trình con giám sát thời gian thực dựa trên Java, nó theo dõi liên tục các tham số của hệ thống và trình bày các thông tin dưới dạng 3D, sử dụng công nghệ VRML và Java EAI. Với công cụ này, một cụm Beowulf lớn có thể thể hiện rõ hiệu quả trong sử dụng một mạng với băng thông rất thấp.

Thực tế, KCAP là một phần mềm mở và có sẵn để dùng bởi tất cả người sử dụng.

8.2 Khái niệm – Vận dụng công cụ quản lý tài nguyên

Tiếp theo khái niệm kiến trúc của công cụ quản lý tài nguyên trong bối cảnh của BKluster. Giới thiệu cơ chế của sự vận hành và các thành phần của công cụ này.

Cuối cùng, phần cuối là sự vận dụng cụ thể và sự tích hợp công cụ quản lý tài nguyên trong hệ thống phân cụm này.

8.2.1 Kiến trúc của hệ thống BKluster

Công cụ quản lý tài nguyên là một phần chính của hệ thống BKluster. Kiến trúc của nó có 3 lớp chính minh họa như hình dưới.

Lớp core

Lớp này là lớp thấp nhất đóng vai trò một nền tảng của hệ thống. Nó bao gồm hệ điều hành (Linux - BProc), thư viện lập trình song song (LAM-MPI) và một vài phần mềm mở cần thiết cho một cụm như PBS, Ganglia…

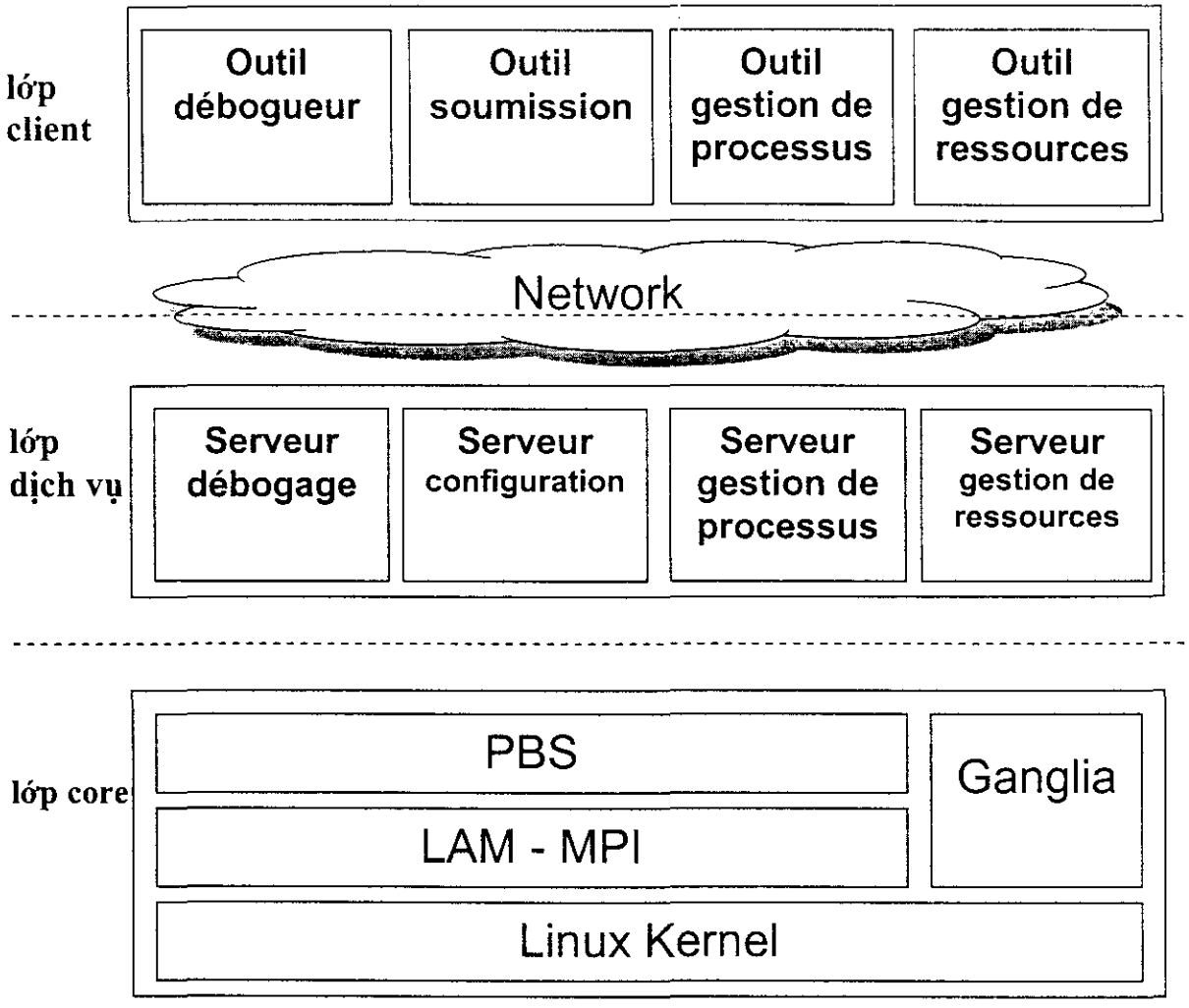
Các thành phần của lớp tim được cài đặt ở tất cả các nút tính toán.

Lớp dịch vụ

Lớp dịch vụ là các lớp ngầm, chúng cung cấp các dịch vụ cần thiết đến lớp client phía trên. Các tiến trình ngầm chạy trên nút Master. Nó chịu trách nhiệm gắn kết 2 lớp thấp nhất và cao nhất của hệ thống.

Lớp client

Lớp này cung cấp các ứng dụng/ công cụ với giao diện đồ họa cho người sử dụng có thể làm việc trên hệ thống. Các ứng dụng này có thể được cài đặt tại tất cả các trạm làm việc (trạm làm việc của cụm này hay một máy tính kết nối với cụm này qua mạng TCP). Người sử dụng và người quản trị thu được sự cho phép làm việc với cụm từ xa (nếu họ có tài khoản trong cụm).



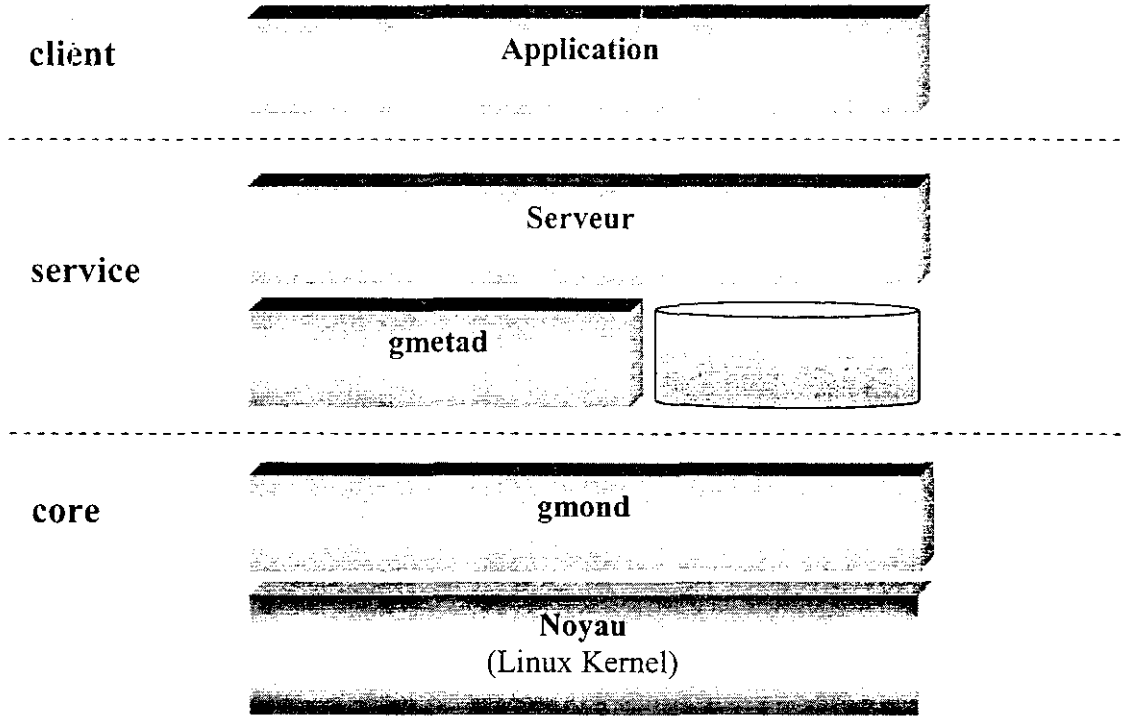
Hình 8-8: Công cụ quản lý tài nguyên trong kiến trúc hệ thống BKluster

8.2.2 Kiến trúc – Cơ chế vận hành của công cụ quản lý

Sau khi có thảo luận chung về kiến trúc và các tài nguyên của hệ thống BKluster, phần này là khái niệm về kiến trúc của công cụ quản lý tài nguyên, cơ chế vận hành và tương tác giữa các thành phần của hệ thống được đề cập. Công cụ này được xây dựng thừa kế cơ chế quản lý tài nguyên của Linux, phần mềm mở Ganglia và sử dụng khả năng thiết kế đồ họa của RRDTool.

8.2.2.1 Kiến trúc công cụ quản lý

Đầu tiên, để có một phần mềm tốt, phải làm tốt các khái niệm của kiến trúc. Hình dưới giới thiệu các thành phần khác nhau trong hệ thống BKluster.



Hình 8-9 Cấu trúc của công cụ quản lý tài nguyên của cụm

Công cụ theo dõi bao gồm nhiều thành phần trong 3 lớp chính, tự truyền thông qua message XML.

Lớp Core

Lớp này gồm 2 thành phần : Nhân (Linux kernel) và gmond của Ganglia.

- **Nhân** : Tất cả các nút tính toán sử dụng hệ điều hành Linux. Hạt nhân Linux quản lý và theo dõi các hoạt động cơ sở, cũng như các tài nguyên của mỗi nút. Các thành phần riêng biệt của công cụ quản lý tài nguyên sử dụng các thông tin tài nguyên qua nhân của Linux ở mỗi nút tính toán để quản lý toàn thể hệ thống.
- **Gmond** : Gmond được xây dựng dựa trên khả năng quản lý tài nguyên của hệ điều hành Linux. Gmond ở các nút tính toán có trách nhiệm thu thập thông tin tài nguyên được theo dõi bởi nhân Linux (Red Hat 9/Fedora). Ở mỗi nút, phương thức thu thập bởi giao diện hệ thống file /proc được sử dụng. Các thông tin thu được này được gửi đến lớp dịch vụ ở trên.

Lớp dịch vụ

Lớp này có 2 tiến trình ngầm : gmetad của Ganglia, server và phần mềm cơ sở dữ

liệu RRDTool.

- **Gmetad**: Là tiến trình thu thập các thông tin của tất cả các nút của hệ thống. Các thông tin này được gửi đi bởi thành phần gmond của lớp dưới. Nó bảo taòn chúng trong các file Round Robin .rrd và nó gửi các thông tin này đến server.
- **RRDTool** : Cơ sở dữ liệu Round Robin .rrd được chọn để sử dụng. RRDTool được sử dụng để lưu các giá trị quan trọng nào đó trong file cơ sở dữ liệu Round Robin. Các giá trị này thay đổi và cập nhật với thời gian. Cơ sở dữ liệu Round Robin được sử dụng trong các ứng dụng thời gian thực. Nó cho phép xây dựng đồ họa.
- **Server** : Server đọc các thông tin của gmetad, nó chọn một số thông tin cần thiết và gửi đến application của lớp trên.

Lớp client

Lớp này là lớp cao nhất là công cụ quản lý tài nguyên với một giao diện đồ họa.

Application : Là giao diện đồ họa cho người sử dụng. Tùy theo thông tin gửi bởi server và dữ liệu trong file Round Robin .rrd, nó hiển thị chúng rõ ràng cho người sử dụng và nhà quản trị hệ thống. Các ảnh hiển thị là được tạo bởi RRDTool.

Truyền thông giữa các thành phần này thông qua chuẩn XML.

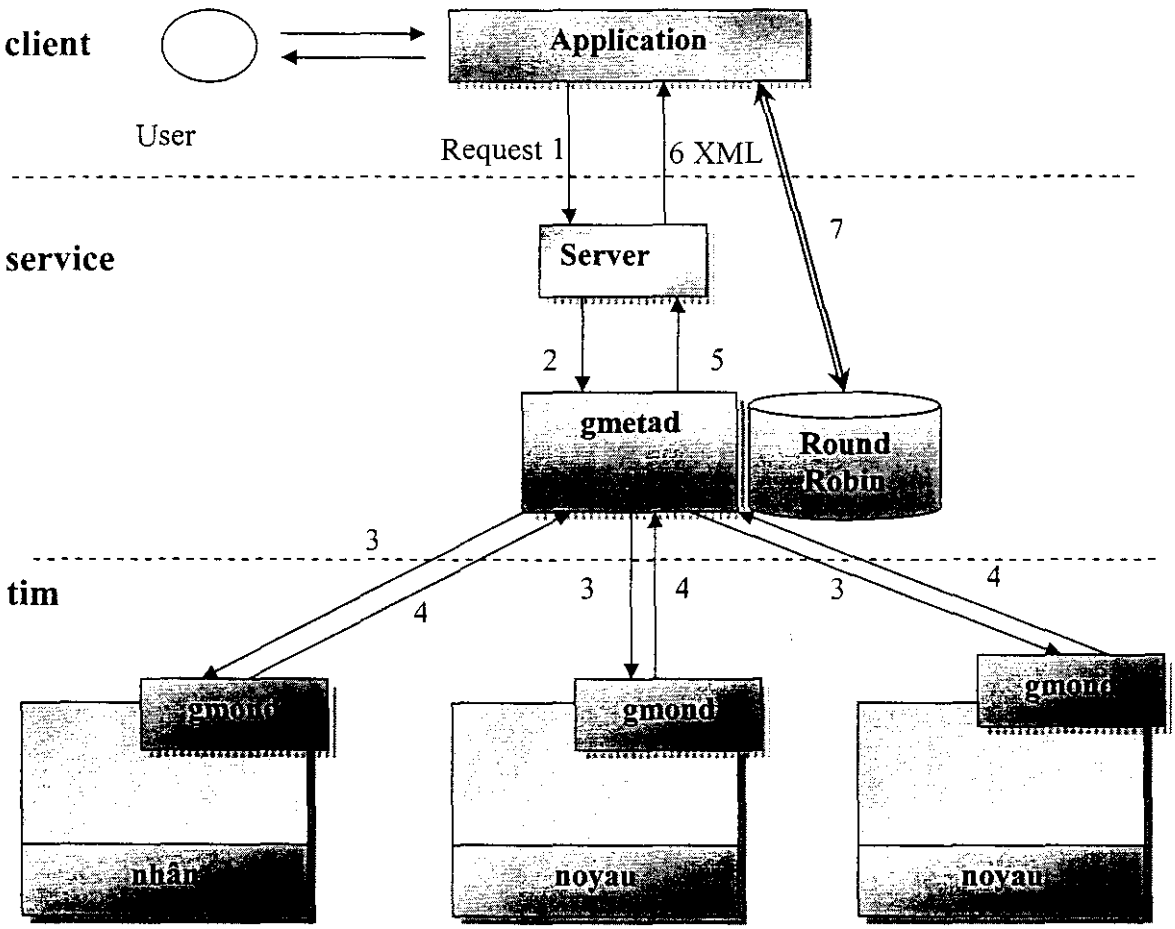
8.2.2.2 Cơ chế vận hành

Cơ chế vận hành bao gồm 7 bước, trong đó bước đầu và bước thứ 6 được thực hiện trong lớp client và service. Bước 3 và 4 thực hiện trong tầng service và core. 3 bước còn lại (2,5,7) trong cùng 1 tầng (tầng service).

- Bước 1 : Application kết nối với server của tầng service và phát tín hiệu yêu cầu đọc
- Bước 2 : Server kết nối với gmetad và phát tín hiệu yeu cầu đọc gmetad.
- Bước 3 : gmetad của tầng service kết nối với gmond của tầng tim và phát tín hiệu yêu cầu đọc tất cả các gmonds.
- Bước 4 : gmond thu thập thôn gtin , gửi chúng đến tầng service.
- Bước 5 : gmetad thu thập thông tin của tất cả các gmonds, bảo vệ chúng

trong các file cơ sở dữ liệu Round Robin .rrd và gửi chúng đến server.

- Bước 6 : Server đọc các dữ liệu gửi đến bởi gmetad, phân tích chúng và gửi thông tin cần thiết đến application .
- Bước 7 : application đọc các dữ liệu nhận bởi server của tầng service. Phân tích nó, đọc các file cơ sở dữ liệu, xây dựng graphic, hiển thị các thông tin cần thiết trong giao diện người sử dụng.



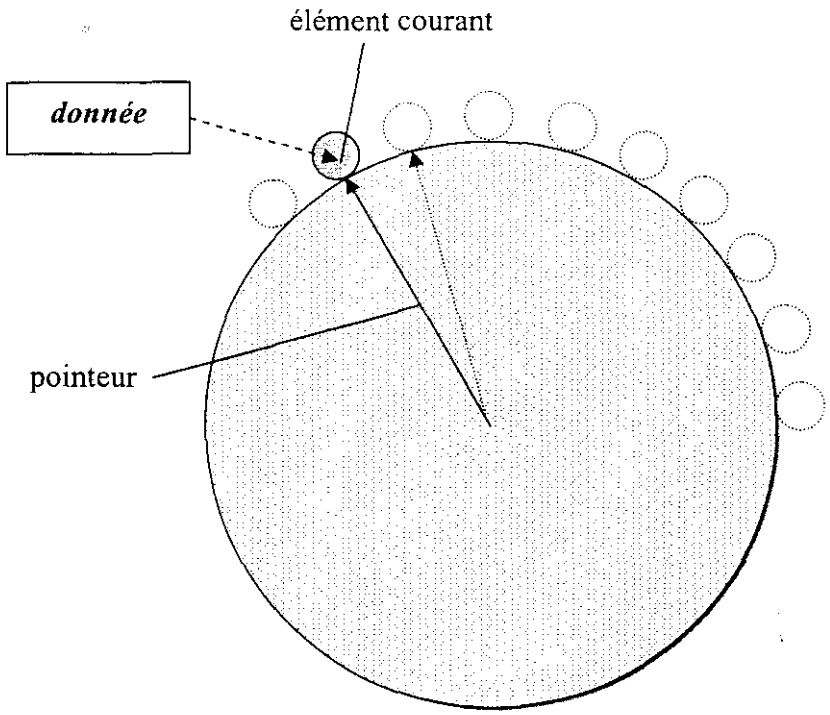
Hình 8-10:Cơ chế vận hành của công cụ quản lý tài nguyên

8.2.3 Cơ sở dữ liệu Round Robin và RRDTool

Round Robin là một cơ sở dữ liệu với lĩnh vực áp dụng chính là lưu cac giá trị thay đổi theo thời gian. RRDTool là một công cụ có khả năng tạo nhanh các hiển thị graphic các giá trị của dữ liệu thu thập được trong một chu kỳ thời gian.

8.2.3.1 RRDTool là gì ?

RRDTool là thành phần của cơ sở dữ liệu Round Robin. Round Robin vận hành với một lượng cố định các dữ liệu, và một con trỏ. Có thể coi Round Robin như một vòng tròn với vài điểm đường nét trên biên các điểm này là nơi mà các dữ liệu có thể được lưu. Đường nối từ trung tâm vòng tròn đến các điểm, đó là con trỏ.



Hình 8-11 Phần tử dữ liệu trong RRDTool

Khi các dữ liệu được đọc hay ghi, con trỏ di chuyển đến phần tử tiếp theo. Vì trên một vòng tròn không cso điểm bắt đầu, và kết thúc, có thể lặp mãi mãi. Sau một thời điểm, tất cả các điểm có thể dùng được đã được sử dụng và tiến trình tái tạo tự động các điểm cũ.

RRDTool vận hành với cơ sở dữ liệu Round Robin (RRDs), là các file nhị phân « .rrd ». Cách lưu dữ liệu trong file .rrd rất đơn giản : đầu tiên, chúng ta đưa một tên , sau đó một giá trị với nhiều thời điểm có thể được đo và cung cấp các thông tin đến RRDTool.

8.2.3.2 Các lệnh chính của RRDTool

Trong RRDTool, tất cả các tương tác giữa cơ sở dữ liệu Round Robin và người sử dụng thôn gqua dòng lệnh. Phần sau sẽ giới thiệu các lệnh chính của công cụ này: các lệnh để tạo một cơ sở dữ liệu, lưu các giá trị, hiển thị các thông tin, tạo một graphic ...

- rrdtool create Tạo một cơ sở dữ liệu mới .rrd
- rrdtool update Cập nhật cơ sở dữ liệu, lưu các giá trị mới trong file .rrd.
- rrdtool graph Tạo một graphic các dữ liệu được lưu trong một hay nhiều file .rrd. Để xay dựng graphic, các dữ liệu cũng có thể khai thác ở stdout.
- rrdtool dump Lấy ra nội dung ASCII của.rrd.
- rrdtool restore Tổ chức lại một.rrd từ định dạng XML đến một.rrd nhị phân .
- rrdtool fetch Lấy các dữ liệu theo chu kì thời gian của một.rrd.
- rrdtool last Tìm thời gian cuối cùng cập nhật của một.rrd.
- rrdtool info Hiển thị các thông tin trên 1 fficher.rrd.
- rrdtool xport Xuất các dữ liệu tìm từ 1 hay nhiều file .rrd.

Ví dụ

Đầu tiên ta có một ví dụ như sau: Trong một cái ôtô đang chạy, sự thay đổi đồng hồ công tơ là liên tục. Nhưng phải chú ý, RRDTool làm việc với các chuông thời gian đặc biệt của thế giới của UNIX. Chuông thời gian là số giây mà nó trôi đi từ 1/1/1970 UTC. Chuông thời gian này thể hiện thời gian local. Ví dụ, 12h 05 ngày 7/5/2005 ở Việt Nam là tương tự với 920804400.

Temps	Kilomètre	Temps	Kilomètre
12h 05	12345 KM	12h 30	12373 KM
12h 10	12357 KM	12h 35	12383 KM
12h 15	12363 KM	12h 40	12393 KM
12h 20	12363 KM	12h 45	12399 KM
12h 25	12363 KM	12h 50	12405 KM

Hình 8-12 Ví dụ về CSDL Round Robin

Tạo cơ sở dữ liệu Round Robin:

```
rrdtool create test.rrd                \
            --start 920804400          \
            DS:speed:COUNTER:600:U:U   \
            RRA:AVERAGE:0.5:1:24       \
            RRA:AVERAGE:0.5:6:10
```

```
Stocker les values de données dans la base de données :
```

```
rrdtool    update    test.rrd    920804700:12345    920805000:12357
920805300:12363
```

```
rrdtool    update    test.rrd    920805600:12363    920805900:12363
920806200:12373
```

```
rrdtool    update    test.rrd    920806500:12383    920806800:12393
920807100:12399
```

```
rrdtool update test.rrd 920807400:12405
```

Hiển thị graphic

Một đặc trưng quan trọng của RRDTool là khả năng tạo graphic. RRDTool tạo một cơ sở dữ liệu, lưu các dữ liệu trong đó, phân tích các dữ liệu này và tạo các graphic trong định dạng GIF/PNG để hiển thị các ảnh trên. Các ảnh này phụ thuộc các dữ liệu thu thập được. Nó có thể được sử dụng để cho thấy sự sử dụng trung bình mạng, nước thủy chiều, tia sáng mặt trời, sức mạnh năng lượng, số khách trong một cuộc triển lãm, các mức tiếng ồn gần sân bay, nhiệt độ trong tủ lạnh , .. Nó phải có bộ dò tìm để đo các dữ liệu và để lưu với RRDTool. Lệnh « graph » sử dụng trong lệnh « fetch » để tìm các giá trị của cơ sở dữ liệu. Với các giá trị tìm được, nó vẽ graphic như định nghĩa bởi các tham số cung cấp trên dòng lệnh. Một graphic đơn giảm có thể đưa ra các DS (sources de données) khác nhau của một cơ sở dữ liệu . Nó có thể đưa ra các giá trị của hơn một cơ sở dữ liệu trong một graphic đơn giản. Thường, nó cần thiết chạy vài phép toán trên các giá trị tìm được của cơ sở dữ liệu trước khi vẽ.

8.2.4 Server

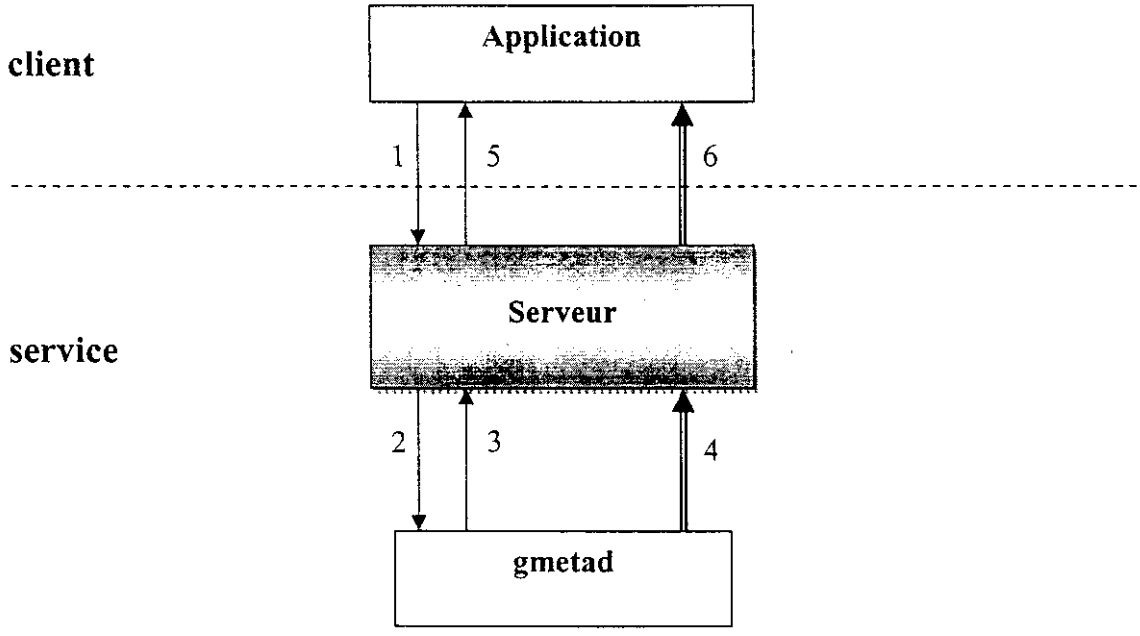
Như đã nói trong phần trước, tiến trình ngầm gmetad thu thập thông tin của tất cả các nút tính toán theo chuẩn XML. Mạng kết nối là một phần quan trọng trong cấu trúc hệ thống phân cụm, từ đó, liên kết giữa các nút tính toán ổn định. Trong trường hợp có nhiều người sử dụng cùng lúc, lưu lượng mạng có thể đầy hay vượt

quá lưu lượng băng thông, nếu không, nó chiếm giữ phần lớn băng thông vì các dữ liệu gửi bởi gmetad không nhỏ. Hơn nữa thời gian phản ứng của mạng phải tính sao cho hợp lý. Nó có thể gây ra một số nguy hiểm cho sự vận hành của công cụ quản lý tài nguyên và ảnh hưởng đến hệ thống. Để khắc phục, phải giảm lượng dữ liệu trao đổi.

Giải pháp đưa ra ở đây là một thành phần có tên « server ». Thành phần application của tầng cao nhất không nối trực tiếp với gmetad của tầng này. Trao đổi giữa 2 thành phần của 2 tầng cao nhất và thấp nhất được thực hiện qua tầng dịch vụ ở giữa. Server như một bộ lọc cho phép các thông tin cần thiết qua. Với giải pháp này, lưu lượng có thể giảm nhiều lần (khoảng 30 lần).

8.2.4.1 Cơ chế vận hành

Công việc kết hợp 2 thành phần là application và gmetad của 2 tầng khác nhau là trách nhiệm chính của server. Chia hoạt động của server ra 6 bước chính :



Hình 8-13 : Cơ chế vận hành của server

Bước 1, 5, 6 trong tầng client và service còn lại trong cùng tầng service.

- Bước 1 : Server có trách nhiệm nhận các yêu cầu từ application của tầng client.

- Bước 2 : Server thiết lập một liên kết với gmetad bằng cách gửi 1 yêu cầu đến gmetad.
- Bước 3 : gmetad trả lời server. Trả lời có thể là từ chối/lỗi hay đồng ý.
- Bước 4 : Nếu đồng ý, server sẽ đọc dữ liệu XML của gmetad.
- Bước 5 : Server nhận gửi hiệu trả lời đến application. Nếu không có lỗi trong quá trình đọc dữ liệu của gmetad, trả lời sẽ chấp nhận. Nếu không trả lời lỗi.
- Bước 6 : Trong trường hợp tín hiệu trả lời toàn là chấp nhận, application có thể đọc dữ liệu XML của server. Server không gửi các dữ liệu này ngay lập tức đến application của tầng cao nhất. Nó phân tích các dữ liệu đã đọc của gmetad, chọn và rút ra các dữ liệu cần thiết. Cuối cùng, nó chuyển đổi các dữ liệu rút ra được sang XML và gửi đến application.

Cổng mà gmetad dùng để xuất dữ liệu XML là 8652 (mặc định) còn server xuất XML đến application qua cổng 8080 (mặc định).

8.2.4.2 Chức năng chính

Theo cơ chế trên, các chức năng chính của server có thể gồm:

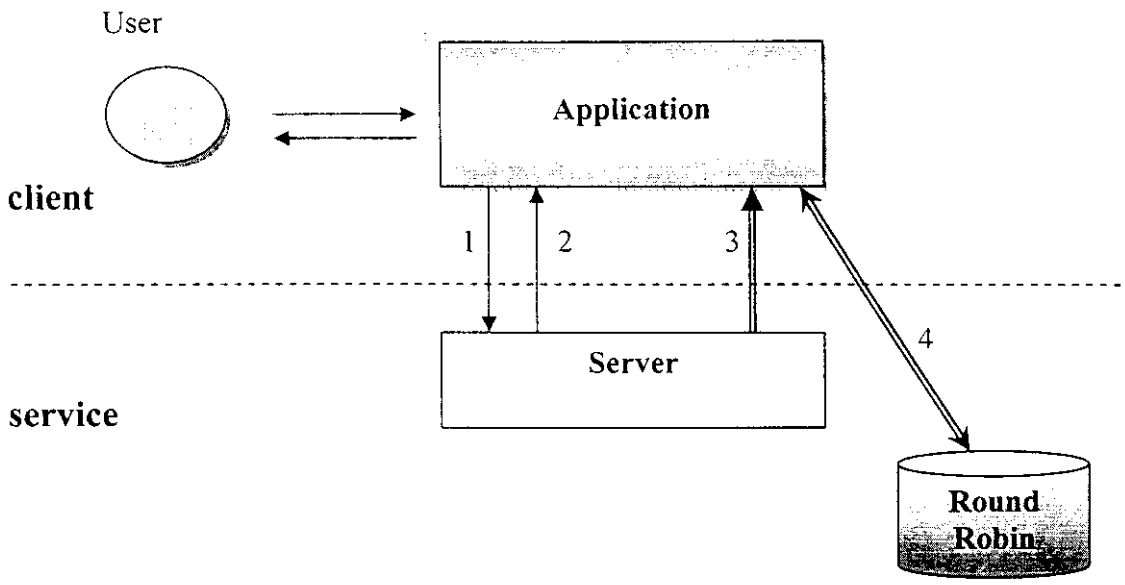
- Nhận và trả lời các yeu cầu của application.
- Đọc dữ liệu XML.
- Phân tích dữ liệu XML.
- Chuyển đổi dữ liệu thường sang XML.
- Gửi dữ liệu XML.

8.2.5 Application

Application là một thành phần của tầng client, cung cấp một giao diện đồ họa cho người sử dụng. Sự truyền thông giữa các phần bên trong trong hệ thống phải trong suốt với người sử dụng.

8.2.5.1 Cơ chế vận hành

Sự vận hành của application của tầng client có thể được minh hoạn qua 4 bước sau:



Hình 8-14 Cơ chế vận hành của application

- Bước 1 : application thiết lập một kết nối với server bằng cách gửi 1 yêu cầu đến server của tầng service.
- Bước 2 : Server gửi tín hiệu trả lời tới application. Có thể là lỗi hay đồng ý. Nó phụ thuộc sự hoạt động của các thành phần bên dưới.
- Bước 3 : Nếu đồng ý, application đọc dữ liệu XML của server. Cổng sử dụng là 8080.
- Bước 4 : Như server, application phân tích dữ liệu XML gửi đi bởi server. Nó sử dụng RRDTool để tạo các graphic từ các file cơ sở sữ liệu Round Robin. Cuối cùng, nó hiển thị các thông tin và ảnh cần thiết cho người sử dụng .

8.2.5.2 Chức năng chính

Từ cơ chế vận hành trên, có thể có một tổng kết về chức năng chính của thành phần này:

- Đọc DL XML.
- Phân tích DL XML
- Dùng RRDTool tạo graphic.
- Hiển thị thông tin và ảnh cần thiết cho người sử dụng.

Thêm vào đó, có thể đọc, công cụ quản lý tài nguyên là một sự kết hợp giữa RRDTool (Round Robin Data base Tool) và một ngôn ngưc lập trình hướng đối tượng trên Linux (Qt). RRDTool có thể quản lý CSDL, và, nó có thể xây dựng các grapgic. Thành phần application sử dụng khả năng này để hiển thị các ảnh cụ thể và rõ ràng.

8.2.6 Vận dụng

Phần này giới thiệu các công việc vận dụng trong công cụ quản lý tài nguyên. Đầu tiên, phải tìm hiểu kiến thức về môi trường pháp triển. Công cụ quản lý được phát triển trên nền Linux bằng việc dùng ngôn ngữ lập trình Qt, Ganglia, RRDTool và chuẩn XML. Cuối cùng, phần quan trọng của chương này là vận dụng cụ thể các thành phần chính của quản lý tài nguyên cụm.

8.2.6.1 XML

Sự truyền thông giữa các thành phần trong hệ thống theo chuẩn XML. XML (EXtensible Markup Language) là một định dạng text đơn giản và mềm dẻo SGML (Standard Generalized Markup Language) (ISO 8879). XML là một nền cross-platform, một phần mềm và một công cụ độc lập với phần cứng để trao đổi thông tin. Như HTML được diễn đạt để trình bày DL và để tập trung sự chú ý lên DL thu thập nào đó, nhưng XML được diễn đạt để mô tả các DL và tập trung sự chú ý lên DL này. Ngày nay, XML có vai trò ngày càng quan trọng trong trao đổi DL lớn, đa dạng trên web và các chỗ khác.

XML được diễn đạt để chuyển các DL. Nhưng XML không thay thế HTML vì XML chưa được diễn đạt để thực hiện. XML được viết cho cấu trúc, lưu và gửi thông tin. Ví dụ:

```
<Patient>
    <PatientName>John Smith</PatientName>
    <PatientAge>108</PatientAge>
    <PatientWeight>155</PatientWeight>
</Patient>
```

Khi một văn bản XML được trao đổi, phải có người gửi và người nhận. Nhưng luôn luôn, tài liệu XML này chẳng thực hiện gì cả. Đó là thông tin đúng được phát triển trong các mục tiêu của XML. Ai đó phải viết một phần mềm để gửi, nhận hay đưa ra thông tin.

Trong ví dụ trên, các thành phần cơ sở của 1 tài liệu XML có thể dễ nhận thấy. Các thành phần chủ yếu nhất của XML là các nhân tố, các thuộc tính, và các lời chú giải.

- **Nhân tố**: Các nhân tố được sử dụng để đánh dấu các đoạn của một tài liệu XML. Một nhân tố XML có dạng:

  <Nom_du_élément>Contenu</Nom_du_élément>

  <Nom_du_élément>, </Nom_du_élément> : tag của XML

  Contenu : Nội dung trong tag XML. Nội dung có thể là nội dung bình thường, hoặc là các nhân tố khác

- **Các thuộc tính**: Một thuộc tính là một cơ chế để thêm mô tả thông tin về mọt nhân tố. Ví dụ, trong tài liệu XML của chúng ta, có ý kiến, nếu trọng lượng của patient đo bằng kg. Để biểu thị PatientWeight cho bởi kg, một đơn vị được thêm vào trong phần tử này và giá trị của nó sẽ biểu thị như KG.

  <PatientWeight unit="LB" unit="KG">155</PatientWeight>

- **Chú giải**: Các chú giải là các mô tả trong một tài liệu XML. Các chú giải trong XML sử dụng cùng cú pháp như HTML , Ví dụ:

  <!-- Text de commentaire-->

8.2.6.2 Ngôn ngữ lập trình Qt

Qt là ngôn ngữ lập trình hướng đối tượng trong môi trường KDE/GNOME Linux. Nó là một thư viện lập trình C++ tuyệt vời tương hợp Linux, Windows et Macintosh. Một code có thể chạy trên các nền khác nhau. Qt được cung cấp với số lượng các công cụ bằng dòng lệnh hay đồ họa, dễ dàng trong phát triển các ứng dụng. Qt là 1 ngôn ngữ mạnh, cho phép xây dựng dễ dàng, nhanh chóng các ứng dụng với một giao diện đồ họa bằng cách dùng Qt Designer. Một ưu điểm khác của Qt là khả năng làm việc với tín hiệu và các slot.

Công cụ của Qt

Đây là một số công cụ cung cấp bởi Qt.

- **Các biến môi trường**
  - Để sử dụng Qt với thành công, 3 biến môi trường phải được cài đặt:

QTDIR, PATH và QMAKESPEC.

- Để compile Qt, QTDIR phải định nghĩa để đánh dấu về hướng danh mục sự phân phối Qt (pointer vers le répertoire contenant la distribution Qt).
- Để có thể gọi các công cụ Qt trên dòng lệnh, biến môi trường PATH phải được định nghĩa. Cho vào đơn giản ${QTDIR}/bin vào cuối của PATH.
- Để sử dụng qmake một cách chính xác QMAKESPEC phải được định nghĩa. QMAKESPEC chỉ đến một danh mục các file mô tả môi trường đích của qmake. Qt bao hàm một danh mục các lời gọi mkspecs, ở trong $QTDIR/mkspecs, bao hàm một số các đặc trưng được sắp đặt. Nếu tiến hành tốt một trong các vừa qua chúng ta làm cho thích hợp với nhu cầu, ví dụ một sự cài đặt của x86 trong Linux sử dụng gcc như trình biên dịch sử dụng linux-g++, QMAKESPEC phải được định nghĩa với $QTDIR/mkspecs/linux-g++.

- **Qt Designer**

Qt Designer là một studio lập trình cung cấp với Qt. Qt Designer là một công cụ để diễn đạt và vận dụng các giao diện người sử dụng xây dựng với Qt. Như một người xây dựng giao diện, với nó có thể dễ dàng tạo ra các hội thoại và tùy ý sử dụng các đối tượng như chúng được thấy khi thực hiện. Một giao diện được tạo ra với Qt Designer được đăng kí trong một file có phần mở rộng (.ui) và chứa code XML. Định dạng này không một cách trực tiếp có thể khai thác bởi trình biên dịch C++. Nó là cần thiết để biến đổi file (.ui) này với một công cụ, gọi là uic (User Interface Compiler), nó đọc một file định nghĩa giao diện (.ui) bằng XML được tạo ra bởi Qt Designer và tạo ra một file (.h) và nguồn (.cpp) tương ứng.

Khi so sánh Qt với các công cụ khác để xây dựng giao diện đồ họa cho các ứng dụng, như Visual C++ của Microsoft, có thể hiểu rằng các code của Qt ngắn hơn (một nửa hay một phần ba). Nó cũng là ưu điểm quan trọng của ngông ngữ này.

- **qmake**

  qmake cho phép tạo ra các file « Makefile » tùe file dự án độc lập của nền.

- **uic**

  uic là trình compiler giao diện người sử dụng, nó đọc một file giao diện (.ui) bằng XML tạo bởi Qt Designer và tạo một file tiêu đề và source C++ tương ứng

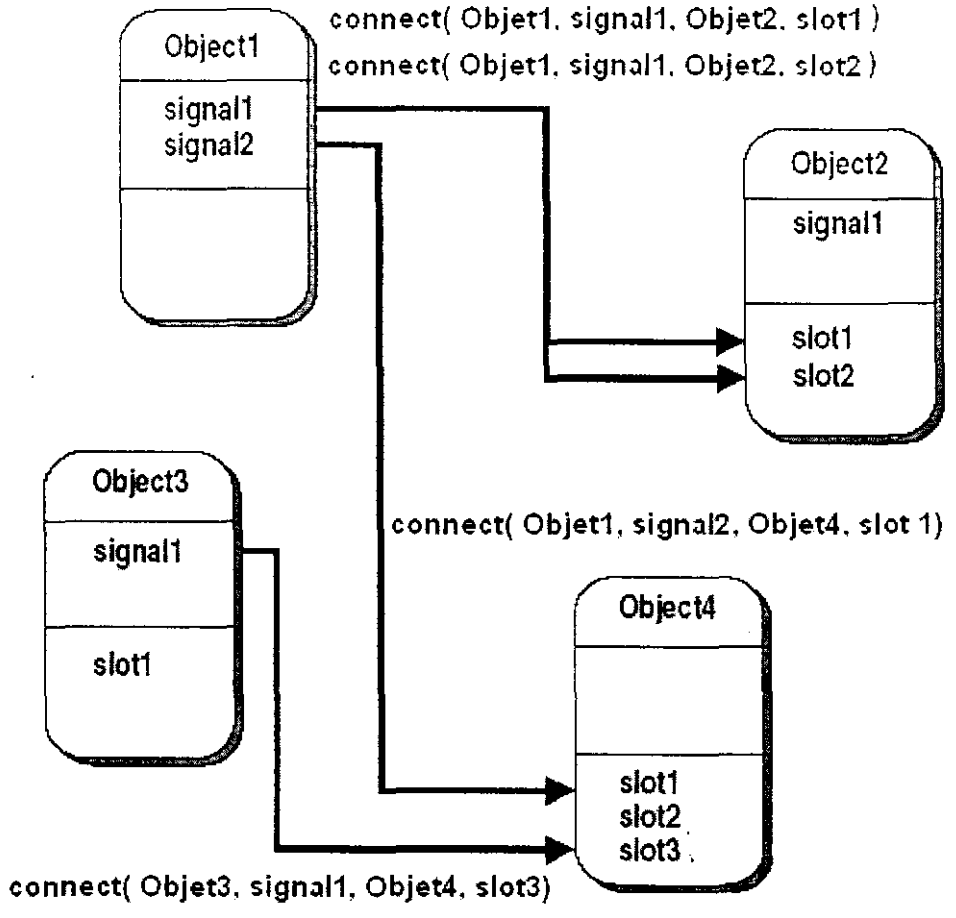
- **moc**

  moc (Meta Object Compiler) là trình compiler méta-objet, là một chương trình thao tác các phần mở rộng C++ của Qt. moc đọc một file source C++. Nếu nó tìm thấy một hay nhiều giá trị kê khai, chứa đựng macro Q_OBJECT, nó sản xuất một file source C++ khác chứa code méta-objet cho các giá trị mà chúng sử dụng macro Q_OBJECT. Mặt khác, code méta-objet được yêu cầu bởi cơ chế signal/slot, và hệ thống đặc tính động.

- **Signal và slot**

  Phần quyết định trung tâm của mô hình đối tượng Qt là một cơ chế rất mạnh cho sự truyền thông của các đối tượng yếu ghép đôi được gọi là tín hiệu (signal) và slots. Có nghĩa rằng phát một tín hiệu không biết đối tượng nào sẽ lấy nó vào tài khoản (nó có thể bị bỏ qua). Với cùng cách, một đối tượng chặn một tín hiệu không biết đối tượng khác nào phát ra tín hiệu. Slot có thể có một hàm của một đối tượng Qt hay một hàm

định nghĩa trong một đối tượng của người lập trình. Công nghệ này cho phép nối liền giữa 2 đối tượng khác kiểu và đưa ra một sự mềm dẻo lớn để phát triển



Hình 8-15: **Các kết nối giữa signal và slot**

- **XML**

Các công việc với các DL XML, một phần lớn trong bước tiến của xử lý DL của công cụ quản lý tài nguyên của cụm. Qt đề xuất một nền XML như module, và người sử dụng có thể không có lợi, nhưng sự xuất bản tự do và các công ty xuất bản có. Qt đưa ra 2 cách tương tác với nội dung XML: DOM và SAX. SAX là đơn giản nhất, để đọc và phân tích . DOM đọc file XML hoàn toàn trong một cây bộ nhớ. Cây này có thể

được đọc và thao tác trước khi được để dành hay ghi mới lên đĩa.

SAX thể hiện sự thay đổi khó khăn các DL cung cấp. Bù lại, nó yêu cầu một không gian bộ nhớ rất ít, có lợi như DOM trong nhiều tình huống.

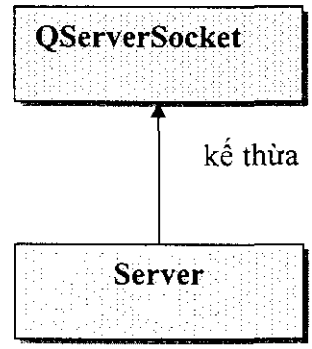
DOM yêu cầu nhiều bộ nhớ hơn. Toàn bộ văn bản phải lưu trong bộ nhớ. Bù lại, nó đưa ra khả năng thay đổi và làm việc với văn bản tự do trong bộ nhớ và đưa cái mới vào đĩa. Nó rất dễ sử dụng trong nhiều tình huống. Qt cung cấp vài lớp để giúp đỡ người lập trình viết và đọc tài liệu XML sử dụng DOM như QDomDocument, QDomElement, QDomNode, QDomAttr...

8.2.6.3 Sự vận dụng

Server

Như trong khái niệm của công cụ quản lý tài nguyên, server là một thành phần nhận các yêu cầy của application. Thành phần này được xây dựng như một tiến trình ngầm để nối 2 thành phần gmetad và application. Tiến trình server ở lớp « Server » thừa kế một lớp sẵn có của Qt: classe QServerSocket.

QServerSocket cung cấp một server dựa trên TCP. Nó là một lớp thích hợp để chấp nhận các liên kết mới của TCP. Người ta có thể đặc tả cổng, và nghe các địa chỉ chính xác hay tất cả các địa chỉ của máy. QServerSocket rất đơn giản để sử dụng: tạo lớp con của nó, gọi hàm của trình xây dựng, và vận dụng hàm newConnection() để đạt được và đáp lại các liên kết mới.




```
class server : public QServerSocket
{
    Q_OBJECT
    private:
```

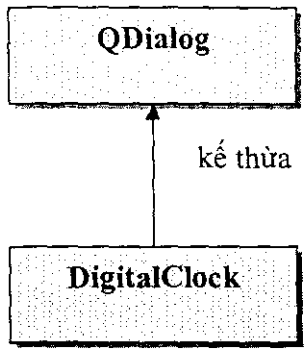
```
        QDomDocument xmlDom;
        QSocket *sock;
        ...
public:
        server( QObject* parent=0 ); //constructeur
        void newConnection( int socket );//nouvelles connexions
signals:
        void newConnect();
        void endConnect();
        void wroteToClient();
        void xmlEndConnect();
        void xmlNewConnect();
        void xmlError(int e);
        void xmlSigConnected();
        void xmlRead();
private slots:
        void xmlConnected();
void readClient();
        void discardClient();
        void xmlEmitError(int e);
        void xmlConnectionClosed();
void readXML();//lire les données XML
  //analyser le document XML
                        //convertir les chaînes de caratère en
forme XML
                    //envoyer le document XML
        void xmlReadyRead();
}
```

Application

Thành phần này có lớp « DigitalClock » nó kế thừa một lớp của Qt: class QDialog

– class cơ sở của tất cả các đối tượng của Qt sử dụng các signal và slot.

Classe QDialog là class cơ sở cho các cửa sổ hội thoại. Một cửa sổ hội thoại được sử dụng cho các công việc thời gian ngắn và các truyền thông ngắn gọn với người sử dụng. QDialog đưa ra các cơ chế như dạng thức, các nút mặc định, khả năng dẫn và một giá trị kết quả.



Cửa sổ giao diện của công cụ quản lý tài nguyên được cập nhật sau mỗi phút. Đó là tại sao , class DigitalClock sử dụng một bộ đếm thời gian (timer).

```
class DigitalClock : public QDialog
//digital clock widget
{  Q_OBJECT
public:
// constructeur
DigitalClock( QWidget * parent = 0, const char * name = 0,
bool modal = FALSE, WFlags f = 0 );
QSocket *sock;
QDomDocument xmlDom;
...
protected:
        void timerEvent( QTimerEvent * );
private slots:
void readXML(); //đọc các DL XML
void xmlReadyRead();
void xmlParse(); //phân tích tài liệu XML
```

```
void update(); //cập nhật giao diện người sử dụng
void getContext();
 private:
   int normalTimer;
}
```

8.3 Kết quả đạt được và hướng phát triển

8.3.1 Kết quả thu được

Công cụ được phát triển là thích hợp với hệ thống BKluster của trung tâm tính toán hiệu năng cao trường đại học Bách Khoa Hà Nội. Nó thỏa mãn các yêu cầu cần thiết khi hệ thống phân cụm được xây dựng.

Cơ sở phần cứng của các thí nghiệm của trung tâm tính toán hiệu năng cao :

Server : Đó là một nút của cụm nhưng nó chịu trách nhiệm quản lý Cluster.

- HP NetServer LH6000
- 6 x CPU Intel Pentium III Xeon 700 MHz
- Bộ nhớ: 1GB SDRAM
- Network Card : 10/100 TX NIC

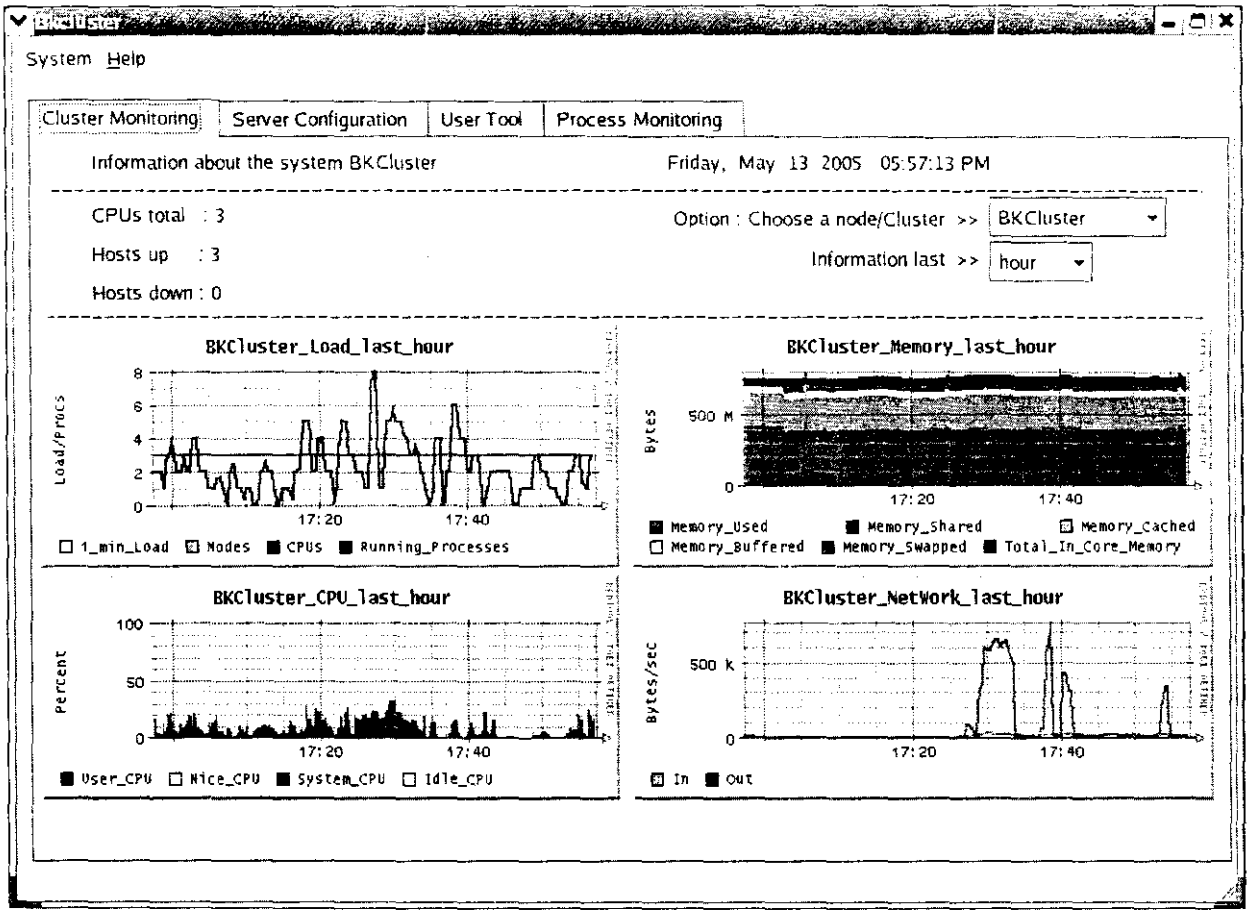
Nút tính toán:

- CPU Intel Pentium III 500 MHz
- Bộ nhớ: 256 MB SDRAM
- Network Card : 10/100 3COM Ethernet Adapter

Switch: BayStack 450-24T 10/100/1000 Switch

Người ta quản lý cụm bằng 2 mức : mức của cụm và mức của các nút.

Về mức cụm



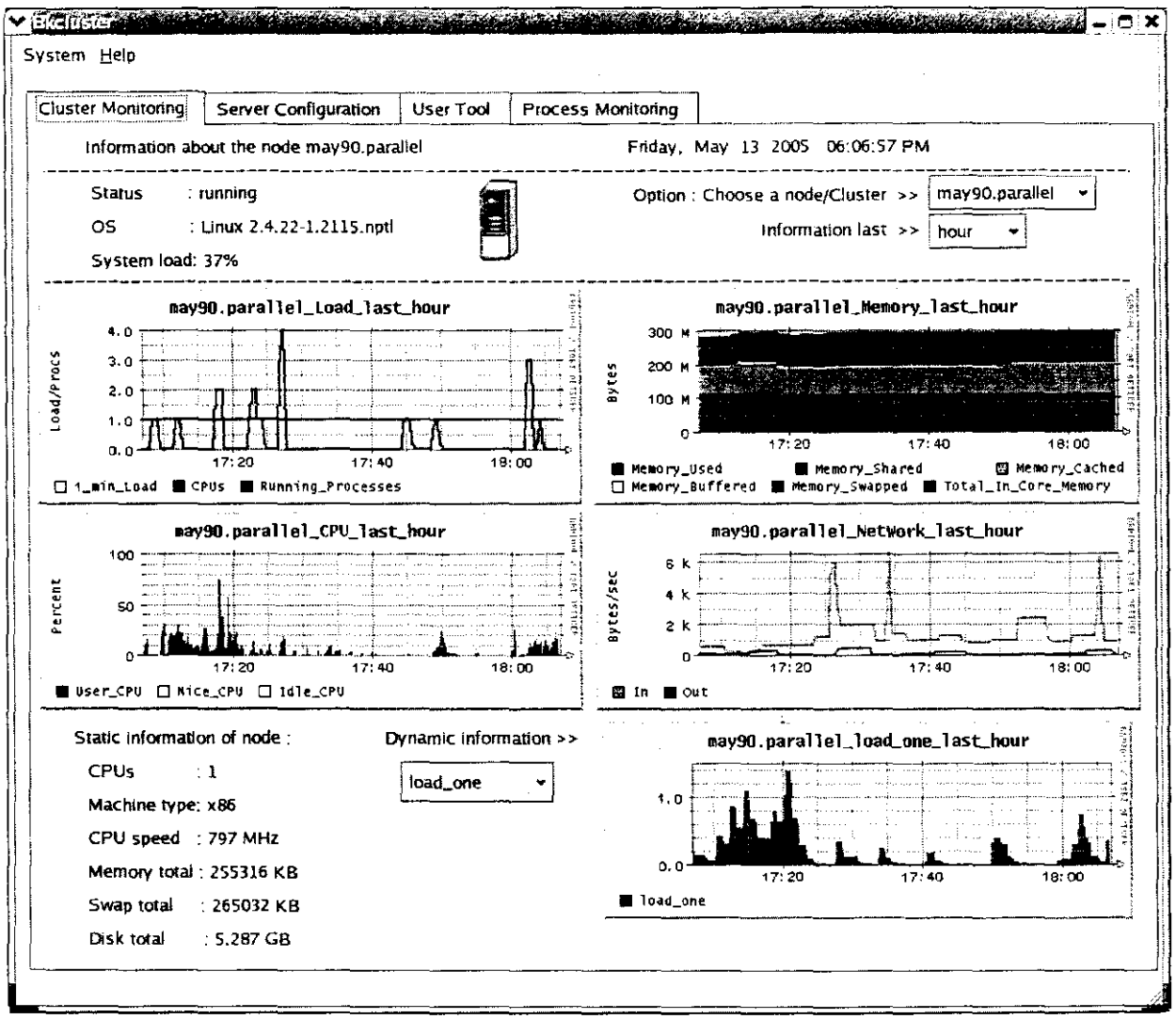
Hình 8-16 Quản lý mức cụm

Người sử dụng, người quản trị cụm hay bất cứ ai có thể xem thông tin chung của hệ thống BKluster bằng cách sử dụng công cụ này :

- Tổng số các tiến trình
- Tổng số các nút hoạt động
- Số các nút không hoạt động
- Hệ thống nạp của cụm
- Bộ nhớ của cụm : tổng, tự do, chia sẻ ..
- Sự sử dụng processor của cụm
- Số lượng byte vào/ra cụm

Về mức nút

Công cụ này cung cấp khả năng xem thông tin chi tiết của mỗi nút trong hệ thống. Theo mặc định, graphic cho phép nhìn các thông tin trong một giờ trước, nhưng người ta có thể xem thông tin liên quan hệ thống chung hay mỗi nút trong 1 ngày trước, 1 tuần trước, 1 tháng trước hay một năm trước. Nó có thể kiểm tra các graphic của một tham số cụ thể trên một nút.



Hình 8-17: Quản lý ở mức nút

Các thông tin thấy được là các thông tin của phần cứng, phần mềm, các thông tin tĩnh cũng như động. Ví dụ:

- Trạng thái hoạt động hay không của nút.
- Hệ điều hành được sử dụng

- Hệ thống nạp nút.
- Số lượng tiến trình của nút.
- Tốc độ processor.
- Bộ nhớ của nút : tổng, rỗi, chia sẻ …
- Utilisation de processeur du nœud.
- Byte vào/ra

…

CHƯƠNG 9 Module Đánh Giá Hiệu Năng Hệ Thống

9.1 Tổng quan về đánh giá hiệu năng hệ thống máy tính

Đánh giá hiệu năng có thể hiểu một cách đơn giản nhất là quá trình chạy các phần mềm chuyên dụng trên một máy tính đơn hay là cả một hệ thống máy tính, từ việc phân tích thời gian chạy chương trình hoặc những kết quả trả về, ta rút ra những kết luận về tốc độ tính toán, tốc độ truyền thông, khả năng truy cập bộ nhớ ... Nội dung chương này sẽ trình bày những vấn đề chung nhất của việc đánh giá hiệu năng như : mục đích, phân loại và sự phát triển của các loại phần mềm đo hiệu năng

9.1.1 Mục đích của việc đánh giá hiệu năng

Khi triển khai một hệ thống máy tính, một yêu cầu đặt ra đối với người quản trị là phải đánh giá được khả năng của hệ thống về các mặt tính toán, truyền thông. Sự đánh giá này có được dựa trên các kết quả cụ thể phản ánh tốc độ thực hiện các thao tác trên các kiểu dữ liệu, tốc độ gửi và nhận gói tin, tốc độ truy cập bộ nhớ ngoài và bộ nhớ trong. Các kết quả này khi được tổng hợp lại sẽ có thể cho phép dự đoán một phần về hiệu năng của những ứng dụng có thể sẽ được triển khai trong tương lai.

Đối với từng máy tính đóng vai trò là nút tính toán, việc đánh giá hiệu năng được thực hiện với mục đích đưa ra những thông số cụ thể phản ánh tốc độ tính toán, tốc độ truy cập bộ nhớ trong khi thực hiện bài toán trên CPU đơn của máy tính. Từ những thông số cụ thể trên ta có thể đưa những đánh giá về năng lực tính toán cũng như kích thước của bài toán là bao nhiêu thì sẽ phù hợp với nút mạng

Khả năng truyền thông giữa các nút trong mạng đóng vai trò rất quan trọng khi triển khai một hệ thống tính toán. Tốc độ truyền thông chịu ảnh hưởng một cách trực tiếp bởi các cấu hình phần cứng cũng như dung lượng gói tin và tần số gửi gói tin. Từ những thông số đo được khi thực hiện các chương trình đo hiệu năng truyền thông, người quản trị có thể đưa ra những thông số tối ưu về dung lượng gói tin và tần số gửi tin của từng giao thức cụ thể đối với hệ thống tính toán đang triển khai

Quá trình đo hiệu năng còn được sử dụng để đánh giá sự tương thích của hệ thống đối với các thư viện lập trình. Trong trường hợp này, phần mềm đo hiệu năng sẽ

gọi một số hàm tiêu biểu của thư viện, đo thời gian thực hiện các hàm này đối với các kích thước đầu vào cụ thể.

Một trong những mục đích của việc xây dựng hệ thống máy tính là để thực hiện các bài toán đòi hỏi số lượng tính toán rất lớn. Lúc này cả hệ thống sẽ được xem như một máy tính duy nhất thực hiện quá trình chạy chương trình. Các phần mềm đo hiệu năng tính toán song song cho phép người quản trị có được những thông số về tốc độ tính toán đối với những kiểu bài toán, ứng dụng khác nhau. Có những chương trình thiên về truyền thông, có những chương trình cho phép giảm thiểu quá trình truyền thông để tập trung đánh giá năng lực tính toán. Để đánh giá sự tương thích của hệ thống đối với những ứng dụng đang dự định triển khai, các phần mềm đo hiệu năng có dạng ứng dụng mô phỏng (simulation application) được viết ra và sử dụng. Các phần mềm này sẽ mô phỏng ứng dụng trong tương lai ở mức độ nhỏ hơn và đo đạc các thông số về tính toán, truyền thông khi chạy với những giá trị đầu vào thích hợp. Qua các thông số thu được, người quản trị hệ thống sẽ có thể đưa ra những đánh giá đúng đắn, những chỉnh sửa để có thể triển khai hệ thống trong tương lai một cách tối ưu nhất về cả cấu trúc phần cứng cũng như các phần mềm cài đặt

9.1.2 Phân loại các phần mềm đo hiệu năng

Những phần mềm đo hiệu năng ra đời vào đầu những năm 80 với mục đích ban đầu là đo hiệu năng tính toán của những máy tính tuần tự. Những chương trình đo hiệu năng ban đầu thường là một tập các câu lệnh được gọi trong nhiều vòng lặp thực hiện một số lượng lớn các phép toán số học với dữ liệu là số nguyên hoặc số thực. Sau này, các phần mềm đo hiệu năng đã được sử dụng trong việc đo đạc, đánh giá các máy tính đơn hoặc cả hệ thống tính toán dưới nhiều khía cạnh như : tính toán, truyền thông, truy cập bộ nhớ ... Với mỗi mục đích khác nhau sẽ có những phần mềm tương ứng để đánh giá, không có một phần mềm đo hiệu năng nào được viết một cách tổng hợp để đánh giá tất cả các khía cạnh trên. Có hai cách phổ biến khi phân loại các phần mềm đo hiệu năng đó là phân loại theo độ phức tạp của chương trình đo hiệu năng và phân loại dựa trên mục đích đo hiệu năng của chương trình

9.1.2.1 Phân loại dựa trên độ phức tạp của chương trình đo hiệu năng

Việc phân loại dựa trên độ phức tạp của chương trình gắn liền với quá trình phát triển của các loại phần mềm đo hiệu năng. Từ những chương trình đơn giản ban đầu viết để đo khả năng tính toán của một máy tính cá nhân đến những chương

trình phức tạp mô phỏng một hệ thống cụ thể triển khai trên toàn bộ hệ thống tính toán. Dựa trên tiêu chí này, các phần mềm đo hiệu năng có thể được phân chia theo mức độ tăng dần thành 3 loại chính : mã lệnh đơn giản (Synthetic Code), hạt nhân (Kernel) và Mô phỏng ứng dụng (Simulation Application). Các chương trình đo hiệu năng ở mức độ mô phỏng ứng dụng đã có thể cho phép người quản trị dự đoán được sự tương thích của hệ thống, thậm chí có thể cả hiệu năng của hệ thống khi thực hiện các ứng dụng có mục đích cụ thể

Mã lệnh đơn giản : là chương trình thực hiện một tập các phép toán như cộng, trừ, nhân, chia trên tập số thực, các thao tác truy cập bộ nhớ như : đọc, ghi dữ liệu từ bộ nhớ ngoài hoặc các thao tác truyền dữ liệu như gửi gói tin và chờ tín hiệu phản hồi từ máy đích. Những chương trình đo hiệu năng tính toán đầu tiên trên Thế Giới thường được xếp vào loại này. Ngày nay những chương trình này thường đã được cải tiến rất nhiều hoặc trở thành một phần của những chương trình đo hiệu năng phức tạp hơn. Tuy nhiên các chương trình đo hiệu năng truyền thông hoặc truy cập bộ nhớ trong hiện nay vẫn được xếp vào dạng mã lệnh đơn giản

Hạt nhân : Đây là chương trình thực hiện một phần của một ứng dụng cụ thể, Hạt nhân thường được sử dụng trong các gói phần mềm đo hiệu năng tính toán. Chức năng của hạt nhân có thể chỉ đơn giản là thực hiện việc chuyển vị ma trận hay phức tạp hơn như thực hiện biến đổi fourier, nhân ma trận. Đa số các chương tình đo hiệu năng tính toán ngày nay được coi là các hạt nhân.

Mô phỏng ứng dụng: Đây là những chương trình được viết ra để đánh giá sự tương thích của hệ thống đối với một loại ứng dụng cụ thể. Các chương trình loại này thường là một tập hợp của các hạt nhân, giữa các hạt nhân đã có sự tương tác với nhau về truyền thông cũng như kết quả tính toán. Hiện nay các chương trình mô phỏng ứng dụng thường được cung cấp dưới dạng một phần của các gói phần mềm đo hiệu năng của các cơ quan, viện nghiên cứu có uy tín trên Thế Giới. Để chạy được các chương trình loại này thì cần phải triển khai toàn bộ hệ thống một cách tương đối hoàn chỉnh về cấu hình phần cứng, phần mềm.

9.1.2.2 Phân loại dựa trên mục đích của chương trình

Các chương trình đo hiệu năng thường chỉ thực hiện việc đánh giá tốc độ thực hiện của một loại công việc cụ thể cho nên việc phân loại theo mục đích ngày nay rất hay được sử dụng trong các tài liệu tổng quan về đánh giá hiệu năng. Có thể nói rằng, bất cứ khía cạnh nào của một máy tính đơn hay một hệ thống tính toán phức tạp đều tồn tại những chương trình đo hiệu năng chuyên dụng để đánh giá hiệu

năng. Có thể chia ra thành các phần mềm đo hiệu năng với những mục đích sau : đo hiệu năng tính toán, đo hiệu năng truy cập bộ nhớ trong, đo hiệu năng truyền thông, đo hiệu năng của thư viện lập trình, đo hiệu năng truy cập bộ nhớ ngoài

Phần mềm đo hiệu năng tính toán

Những chương trình đo hiệu năng chuyên dụng đo hiệu năng tính toán được ra đời vào đầu thập kỷ 80. Chương trình WhetStone được coi là chương trình đo hiệu năng tính toán đầu tiên trên Thế Giới, đây là một tập các module con, mỗi module thực hiện một thao tác riêng như : các phép toán số nguyên, các phép toán số thực. Kiểu dữ liệu sử dụng trong WhetStone có thể có độ chính xác đơn hoặc chính xác kép. Sau WhetStone, xuất hiện các chương trình đo hiệu năng tính toán khác như : DhryStone, Digital Review. Đặc điểm chung của các chương trình này là đều thao tác chủ yếu trên các kiểu dữ liệu số thực, giá trị trả về có thứ nguyên được quy ước riêng, ví dụ WhetStone trả về giá trị có thứ nguyên KWIPS (Kilo WhetStone Instrution Per Second), DhryStone trả về giá trị có thứ nguyên DhryStone Per Second, Digital Review trả về giá trị có thứ nguyên MVUPS (MicroVAX units of processing). Các giá trị này là số nghìn hay triệu vòng lặp dùng trong chương trình tương ứng, rõ ràng là các giá trị này chỉ có thể được sử dụng để so sánh hiệu năng tính toán của hai máy khi cùng chạy một chương trình đo hiệu năng

Bên cạnh những chương trình đo hiệu năng hoạt động chủ yếu trên kiểu dữ liệu số thực, đã xuất hiện các chương trình đo hiệu năng thao tác trên kiểu dữ liệu số nguyên. Các chương trình này thường được cài đặt theo một số giải thuật cụ thể như : HeapSort, Hanoi ... Giá trị trả về của các chương trình này có dạng MIPS (Mega Instruction Per Second) là số triệu lần các câu lệnh được thực hiện trong 1 giây

Các chương trình đo hiệu năng tính toán được phát triển trong những năm gần đây đều sử dụng chủ yếu các phép toán trên kiểu dữ liệu số thực dấu phẩy động (floating point). Kết quả trả về có thứ nguyên là MFLOPS - số hàng triệu phép tính dấu phẩy động trong 1 giây. Một trong những chương trình tiêu biểu đầu tiên có sử dụng kiểu thứ nguyên này là Linpack của tác giả Jack Dongarra trường đại học Tennesse. Linpack là một tập các chương trình con nằm trong thư viện lập trình Lapack. Đây là chương trình đo hiệu năng có số lượng phép tính dấu phẩy động rất lớn tuy nhiên cũng có một nhược điểm là không thực hiện phép chia. Phần mềm đo hiệu năng này còn được sử dụng rất phổ biến và đã được phát triển thêm các phiên bản viết bằng ngôn ngữ C và Java từ phiên bản ban đầu viết bằng

ngôn ngữ Fortran. Các chương trình đo hiệu năng sử dụng phép toán dấu phẩy động sau này đa số đều được xếp vào dạng hạt nhân, ngoài Linpack ra còn có thêm một số phần mềm đo hiệu năng nổi tiếng khác như : Livermore, Nasa Parallel Benchmark (NPB). Đặc biệt gói phần mềm NPB đã cung cấp 5 hạt nhân, 3 mô phỏng ứng dụng được sử dụng rất rộng rãi trong việc đánh giá hiệu năng của các hệ thống tính toán song song

Phần mềm đo hiệu năng truyền thông mạng

Các chương trình đo hiệu năng truyền thông mạng được sử dụng để đánh giá hiệu năng truyền gói tin giữa hai máy đóng vai trò nút mạng trong một hệ thống tính toán. Các chương trình này đều có cấu trúc client – server với module client được cài trên máy nguồn, module server được cài trên máy đích. Máy nguồn sẽ gửi gói tin đến máy đích, tuỳ theo từng chương trình mà máy đích có gửi lại tín hiệu hay không. Người sử dụng có thể tuỳ chọn nhiều giao thức truyền thông như : TCP, UDP hoặc thay đổi giá trị của các gói tin gửi đi, thay đổi độ lớn của bộ đệm socket máy gửi và máy nhận ... để có thể khảo sát tốc độ truyền thông giữa hai máy trong hệ thống dưới nhiều góc độ. Dưới đây là một số phần mềm đo hiệu năng truyền thông mạng phổ biến :

- NetPerf của hãng Hewllet – Packard, phần mềm này được viết vào năm 1996, đến nay đã có phiên bản 3.0
- Iperf của nhóm tác giả tại đại học bang Illinois, phần mềm này được công bố lần đầu vào năm 1999, đến nay đã có phiên bản 1.7.0
- UDPMoN của tác giả Richard Hughes-Jones, phần mềm này được viết vào năm 2000, đến nay đã có phiên bản 3.2

Các chương trình trên đều được viết bằng ngôn ngữ C, trong các phần mềm trên, phần mềm Netperf hay được dùng nhất khi đánh giá hiệu năng truyền thông trong mạng LAN

Phần mềm đo hiệu năng truy cập bộ nhớ trong

Các chương trình đo hiệu năng truy cập bộ nhớ trong không có nhiều như những chương trình đo hiệu năng tính toán hay đo hiệu năng truyền thông mạng nhưng đóng vai trò quan trọng khi phân tích sự tương quan của dung lượng của bộ nhớ trong với tốc độ của CPU với hiệu năng tính toán. Các chương trình này sẽ tiến hành đo thời gian, tần suất truy cập bộ nhớ trong khi thực hiện các phép toán đối với các bộ giá trị đầu vào có kích thước khác nhau. Các chương trình đo hiệu năng

truy cập bộ nhớ trong sẽ cho người quản trị biết được kích thước của bài toán như thế nào là tối ưu nhất đối với từng máy tính của hệ thống. Phần mềm đo hiệu năng truy cập bộ nhớ trong phổ biến nhất là Stream của tác giả John McCalpin, giảng viên trường đại học Delaware, ngoài ra còn các phần mềm khác như Sandra Memory Benchmark, CacheBench ... Thực tế cho thấy kết quả của các phần mềm đo hiệu năng truy cập bộ nhớ trong thường đưa ra thông số tối ưu nhỏ hơn các thông số của nhà cung cấp thiết bị phần cứng

Phần mềm đo hiệu năng của thư viện chương trình

Khi xây dựng các thư viện chương trình, một trong những yêu cầu đặt ra là phải kiểm tra các hàm trong thư viện có hoạt động tốt, tương thích với hệ thống, tốc độ thực hiện ... Tuy nhiên, số chương trình đo hiệu năng của thư viện lập trình là không nhiều, một số đại diện cho những phần mềm thuộc loại này là :

- ATLAS đo hiệu năng của thư viện tính toán BLAS
- Linpack đo hiệu năng của thư viện tính toán Lapack
- NetPIPE đo hiệu năng của thư viện truyền thông điệp MPI. Về bản chất NetPIPE là chương trình dùng để so sánh hiệu năng truyền thông giữa một số giao thức, tuy nhiên NetPIPE thường được cấu hình khi biên dịch để đánh giá hiệu năng của môi trường truyền thông thiết lập bởi thư viện MPI trong các hệ thống tính toán song song ghép cụm

Phần mềm đo hiệu năng truy cập bộ nhớ ngoài

Quá trình đo hiệu năng của bộ nhớ ngoài có thể xem là việc đánh giá sự đáp ứng của hệ thống đối với các lệnh đọc, ghi, khởi tạo hay xoá : creat, read, write, mkdir ... Đối tượng tác động của các câu lệnh này là data hoặc meta-data. Data là các file chứa dữ liệu thông thường còn meta-data là các file chứa dữ liệu về cấu trúc của hệ thống. Meta-data có thể là inodes, free map, indirect blocks và directory.

Trong các hệ thống máy tính, bộ nhớ ngoài tồn tại chủ yếu dưới hai dạng sau : bộ nhớ trên máy trạm và hệ thống file mạng Network File System – NFS. Ứng với mỗi loại bộ nhớ ngoài trên sẽ có các phần mềm đo hiệu năng tương ứng. Dưới đây là một số các phần mềm đo hiệu năng truy cập bộ nhớ ngoài thông dụng :

Các phần mềm đo hiệu năng của bộ nhớ trên máy trạm : IOStone, Bonnie++, IOZone trong đó IOStone được coi là chương trình đo hiệu năng truy cập bộ nhớ ngoài đầu tiên trên Thế Giới

Các phần mềm đo hiệu năng truy cập hệ thống file mạng : NFSStone, NHFStone

Hầu hết các chương trình đo hiệu năng thuộc loại này được viết bằng ngôn ngữ C hoặc C++.

9.1.3 Sự phát triển của các phần mềm đo hiệu năng

Các chương trình đánh giá hiệu năng đầu tiên xuất hiện vào những năm đầu thập kỷ 80 và được áp dụng cho những kiến trúc máy tính, hệ thống tính toán cụ thể, sau đó một số chương trình được phát triển để có thể đo hiệu năng của những hệ thống máy tính khác. Hiện nay, các chương trình đo hiệu năng đã phát triển rất phong phú về số lượng và được áp dụng để đánh giá máy tính đơn hoặc cả hệ thống về nhiều mặt. Quá trình phát triển của các phần mềm đo hiệu năng có thể được chia ra thành hai khía cạnh chính sau : sự phát triển các chương trình để có thể đo hiệu năng của nhiều hệ thống và sự thống nhất thứ nguyên của các chương trình đo hiệu năng

Sự phát triển về khả năng tương thích hệ thống của các chương trình đo hiệu năng: ban đầu, các chương trình đo hiệu năng thường được viết để đánh giá hiệu năng của những hệ thống cụ thể. Điều này được thể hiện qua một số ví dụ như sau:

- Gói phần mềm NASA Parallel Benchmark được viết để đánh giá hiệu năng tính toán của hệ thống tính toán song song ghép cụm của cơ quan nghiên cứu Vũ Trụ Hoa Kỳ với các chương trình hạt nhân và phần mềm mô phỏng dựa trên những yêu cầu nghiên cứu thực tế. Sau này, các chương trình đo hiệu năng trong gói phần mềm NPB đã được hoàn thiện với các module viết bằng ngôn ngữ C, Fortran và High Performance Fortran với các tải đầu vào có kích thước khác nhau. Thực tế cho thấy NPB tương thích với hầu hết các hệ thống tính toán song song trên Thế Giới
- Phần mềm Lipack ban đầu được tác giả Jack Dogarra viết để đo tốc độ thực hiện hai hàm DGEFA và DGESL trong thư viện tính toán Lapack viết bằng ngôn ngữ Fortran. Sau này Lipack đã được sử dụng rộng rãi để đánh giá hiệu năng tính toán của CPU đơn và được phát triển thêm phiên bản viết bằng ngôn ngữ C, gần đây có thêm phiên bản viết bằng ngôn ngữ Java.

Hiện nay đã xuất hiện các công cụ đánh giá hiệu năng được cung cấp dưới dạng các gói. Trong gói bao gồm nhiều chương trình đo hiệu năng với những mục tiêu đo khác nhau. Những gói chương trình đo hiệu năng phổ biến là : NPB - Nasa Parallel Benchmark), ParkBench - Parallel Kernel Benchmark, SPEC - Standard

Performance Evaluation Cooporation, LMBench của hãng Hewllet Packard ...

Sự thống nhất thứ nguyên của các chương trình đo hiệu năng: Các chương trình đo hiệu năng ban đầu sử dụng các thứ nguyên được định nghĩa riêng (như KWIPS, DhryStone Per Second, MVUPS đã được trình bày ở phần 1.2.2). Sau này, hầu hết các chương trình đo hiệu năng gần đây đều sử dụng hai thứ nguyên sau : MFLOPS (số nghìn lần phép tính số thực trong 1 giây) đối với kết quả đo hiệu năng tính toán và Mbps (số nghìn bit truyền thông trong 1 giây) đối với các kết quả đo hiệu năng truyền thông. Sự thống nhất này cho phép xây dựng lên các cơ sở dữ liệu về hiệu năng và khả năng so sánh hiệu năng của các hệ thống máy tính khác nhau trên Thế Giới

9.2 Các phương pháp đo hiệu năng

Nội dung chương này sẽ trình bày về phương pháp xây dựng các phần mềm đo hiệu năng và ý nghĩa của những thứ nguyên của các kết quả đo hiệu năng

9.2.1 Đo hiệu năng tính toán

Để đánh giá hiệu năng của hệ thống, thông thường, cần phải xác định 2 thông số RPEAK và RMAX.

RPEAK: hiệu năng tối đa (hiệu năng thiết kế) của hệ thống. Thông số này được tính bằng tổng năng lực của toàn bộ nút tính toán trong hệ thống và coi như bỏ qua sự tương tác giữa các nút. Công thức tính RPEAK như sau:

```
Rpeak = Số VXL * ClockSpeed * FLOPS/Cycle
1 số thông số FLOPS/Cycle:
PII:1, PIV: 2, Athlon: 3, Italium: 4
```

RMAX: hiệu năng của hệ thống thu được khi thực hiện một chương trình đánh giá hiệu năng chuyên dụng. Phần mềm đánh giá hiệu năng được sử dụng phổ biến nhất trên Thế Giới trong việc xếp loại 500 máy tính mạnh nhất (www.top500.org) là High Performance Linpack (HPL).

Theo ý kiến một số chuyên gia, kết quả RMAX thu được bằng HPL của một số hệ thống tính toán trên Thế Giới thường là 50% RPEAK. Tuy nhiên, tỷ lệ RMAX = 50% RPEAK là đạt được ở những cluster có giải pháp truyền thông rất tốt (sử dụng HPS, Myrinet hoặc InfiniBand), còn với các cluster mà tốc độ truyền thông mạng nhỏ hơn (như Fast Ethernet 100Mbps), tỷ lệ này nhỏ hơn nhiều (ví dụ là 32.7% đối với hệ thống BKluster).

Nguyên tắc hoạt động chung của các chương trình đo hiệu năng tính toán là thực hiện một số lớn các câu lệnh tính toán, đo thời gian chạy toàn bộ chương trình, sau đó tính ra kết quả có dạng số phép tính trong một đơn vị thời gian hoặc tổng số câu lệnh, tổng số hàm thực hiện trong một đơn vị thời gian

Những chương trình đo hiệu năng tính toán ban đầu như WhetStone, DhryStone đo hiệu năng bằng cách thực hiện một tập các module con, mỗi module có chức năng thực hiện phép toán trên các kiểu dữ liệu riêng như : số nguyên, số thực hay thực hiện các câu lệnh như : lệnh rẽ nhánh, lệnh gọi các chương trình con. Với mỗi module, người sử dụng có thể thay đổi các tham số đầu vào cho phù hợp với từng hệ thống cụ thể. Các tham số này liên quan đến kích thước của dữ liệu đầu vào dưới dạng số nghìn lần đoạn lệnh (instruction) thực hiện trong mỗi module. Kết

quả trả về của chương trình sẽ có thứ nguyên được định nghĩa riêng như :

- MWIPS (số nghìn đoạn lệnh của WhetStone trong 1 đơn vị thời gian) đối với các kết quả trả về của chương trình WhetStone
- DhryStone Per Second (số vòng lặp DhryStone trong 1 đơn vị thời gian) đối với kết quả trả về của chương trình DhryStone

Các chương trình đo hiệu năng tính toán từ dạng mã lệnh đơn giản (Synthetic Code) được phát triển lên thành các chương trình (hoặc module chương trình) thực hiện mục đích cụ thể hơn trong tính toán khoa học, đó là các hạt nhân (kernel) và mô phỏng ứng dụng (Simulation Application). Đối với những chương trình ở dạng trên, các phép toán chủ yếu thực hiện trên miền số thực và kết quả được trả về có thứ nguyên là MFLOPS - số nghìn lần phép toán số thực trong 1 giây. Tuy nhiên, tuỳ theo từng mục đích đặc biệt, vẫn có thể tồn tại những thứ nguyên khác (tuy nhiên hiện nay điều này rất ít khi xảy ra), ví dụ như thứ nguyên kết quả trả về của 2 hạt nhân IS (Interger Sort) và EP (Embrassingly Parallel) trong gói phần mềm NPB có thứ nguyên là MOPS - số nghìn lần phép toán trong 1 giây. Đối với hạt nhân IS, phép toán được thực hiện là "sắp xếp số nguyên" còn đối với hạt nhân EP, phép toán được thực hiện là "sinh ngẫu nhiên các số phức".

Đơn vị đo hiệu năng thông dụng nhất hiện nay là FLOPS, để tính ra được kết quả theo thứ nguyên này cần phải xác định thời gian thực hiện chương trình (Benchmark Time) và tổng số phép toán dấu phẩy động (Floating Point Operation Count) được thực hiện.

Thời gian thực hiện chương trình (Benchmark Time) ký hiệu là T(N,P) : được định nghĩa là thời gian để thực hiện bài toán là N (size N) với p vi xử lí. Bài toán được gọi là có độ lớn N nếu vector biểu diễn các tham số đầu vào của bài toán có N phần tử.

Số phép toán dấu phẩy động (floating point operation count) ký hiệu là F(N) : đại lượng này được tính bằng số phép toán dấu phẩy động cần được thực hiện khi giải quyết bài toán có kích thước N. Việc kiểm tra được tiến hành trên cả bài toán đo hiệu năng và chương trình cài đặt tương ứng để xác định ra số phép tính dấu phẩy động cần phải thực hiện trong việc giải quyết các bài toán đơn giản như synthetic và kernel benchmarks. Hơn nữa, có thể còn cần phải có một chương trình đóng vai trò bộ đếm hoặc một phần cứng chuyên dụng dùng để đếm các phép toán dấu phẩy động được thực hiện khi giải quyết các bài toán phức tạp. Để phân biệt các loại phép tính dấu phẩy động, người ta định nghĩa thêm các loại số đo như

sau :

- Benchmark floating point operation count ký hiệu là FB(N) : là số phép toán dấu phẩy động được sử dụng để thực hiện bài toán ở dạng tuần tự
- Hardware floating point operation count ký hiệu là FH(N,p) : là toàn bộ số phép toán dấu phẩy động mà phần cứng phải thực hiện khi giải quyết bài toán ở dạng song song với p vi xử lí. Giá trị này có thể lớn hơn giá trị FB(N)

Dưới đây là bảng biểu diễn số phép toán trên dấu phẩy động (flop) thực sự đối với từng phép tính cụ thể

Kiểu phép toán số thực	Số phép toán số thực (flop)
cộng, trừ, nhân	1
chia, căn bặc hai	4
luỹ thừa, sin, cos ...	8

Bảng 9-1 Số phép toán số thực tương ứng với các phép tính

Từ hai thông số trên ta rút ra được số phép toán dấu phẩy động thực hiện trong một đơn vị thời gian (Benchmark Performance) ký hiệu là RB(N,T) : được định nghĩa bằng tỷ lệ của Benchmark Floating point Operation so với Benchmark Time

$$RB(N,p) = FB(N)/T(N;P)$$

Đơn vị đo của đại lượng này là FLOP/s

Qua công thức định nghĩa trên, ta thấy rằng hiệu năng của hệ thống được đánh giá dựa trên benchmark floating point Operation Count hơn là dựa vào các phép toán dấu phẩy động đã thực sự được phần cứng thực hiện. Mặc dù việc tính toán với bộ nhớ phân tán có thể sẽ thực hiện nhiều phép tính hơn là việc tính toán tuần tự, tuy nhiên chỉ có Benchmark Floating Point Operation Count là được sử dụng để đo hiệu năng của hệ thống

9.2.2 Đo hiệu năng truy cập bộ nhớ trong

Các phần mềm đo hiệu năng truy cập bộ nhớ trong (RAM) thường được xây dựng theo nguyên tắc sau :

- Sử dụng một khối lượng lớn bộ nhớ trong để cấp phát cho các biến tĩnh sử dụng trong chương trình (thường là kiểu mảng 1 chiều với các phần tử số thực)
- Thực hiện các phép toán cộng, trừ, nhân, chia hoặc tổ hợp các phép toán trên, đo thời gian thực hiện cả chương trình, từ đó tính ra kích thước của những dữ liệu đã cấp phát được xử lý trong một đơn vị thời gian

Kết quả trả về của các phần mềm đo hiệu năng truy cập bộ nhớ trong thường có thứ nguyên là MB/s (MegaByte per Second). Thứ nguyên này biểu diễn dữ liệu tính theo MegaByte được xử lý trong vòng 1 giây

Một đặc điểm của hệ điều hành Linux là khi tổng kích thước của bộ nhớ cần cấp phát vượt dung lượng của bộ nhớ trong thì sẽ dẫn đến việc truy cập vào vùng đệm swap. Swap thuộc vào bộ nhớ ngoài nên tốc độ truy cập sẽ chậm hơn tốc độ truy cập vào bộ nhớ trong rất nhiều, điều này sẽ gây ra việc giảm đột ngột tốc độ thực hiện các phép toán trên. Từ các kết quả đo hiệu năng truy cập bộ nhớ trong, người quản trị có thể biết được kích thước cực đại của bài toán là bao nhiêu thì tốc độ tính toán là cao nhất

9.2.3 Đo hiệu năng truyền thông mạng

Các chương trình đo hiệu năng truyền thông mạng luôn có hai thành phần, module server cài đặt trên máy đích và module client cài đặt trên máy nguồn. Module server sẽ gửi các gói tin có kích thước khác nhau đến máy đích, máy đích sẽ đo thời gian từ lúc gửi đến lúc nhận, kích thước của gói tin nhận được, ... Tuỳ theo từng phần mềm mà những kết quả này có thể được hiển thị ngay tại máy đích hoặc được gửi về và hiển thị tại máy nguồn. Để thực hiện điều này, hai module server và client sẽ tạo ra hai kênh thông tin sau :

- **Kênh điều khiển:** kênh này có nhiệm vụ truyền các tham số thiết lập ban đầu, thông tin về cấu hình (khả năng này cho phép dùng module client để thiết lập cấu hình của máy đích) ... Kênh này cũng đóng vai trò truyền kết quả về máy nguồn (nếu cần)
- **Kênh dữ liệu:** kênh này độc lập với kênh điều khiển có chức năng truyền các gói dữ liệu từ máy nguồn đến máy đích để thực hiện phép đo, quá trình kết nối và giao thức truyền thông tuỳ thuộc vào mục đích đo hiệu năng và tuân theo sự thiết lập ban đầu của kênh điều khiển

Tốc độ truyền dữ liệu trên mạng phụ thuộc vào một số yếu tố sau :

- Giao thức gửi và nhận tin
- Kích thước gói tin gửi
- Kích thước bộ nhớ đệm của socket tại máy gửi và máy nhận

Các thông số này có thể được thay đổi khi biên dịch các module hoặc nhập vào dưới dạng tham số khi thực hiện chương trình. Kết quả đo hiệu năng truyền thông mạng thường có thứ nguyên là MBps : MegaBit per second. Đây là tổng số bit truyền được trong 1 giây

9.2.4 Đo hiệu năng của thư viện phần mềm

Đo hiệu năng của thư viện lập trình thực chất là việc đo thời gian thực hiện một số hàm quan trọng trong thư viện với các giá trị đầu vào khác nhau. Tuỳ vào chức năng của thư viện cũng như mục đích của việc đánh giá hiệu năng mà có hai cách đánh giá hiệu năng của thư viện trực tiếp và gián tiếp

Đánh giá hiệu năng thư viện phần mềm một cách trực tiếp : một số chương trình kiểu này là Linpack đánh giá hiệu năng thư viện tính toán Lapack, ATLAS đánh giá hiệu năng thư viện tính toán BLAS. Kết quả trả về có thứ nguyên phụ thuộc vào chức năng của thư viện, ví dụ Linpack trả về kết quả có thứ nguyên là MFLOPS

Đánh giá hiệu năng thư viện phần mềm một cách gián tiếp : Không phải thư viện phần mềm nào cũng tồn tại phần mềm riêng đánh giá hiệu năng. Trong trường hợp này thì quá trình đánh giá hiệu năng có thể thực hiện một cách gián tiếp dựa vào các phần mềm chuyên dụng khác, ví dụ như phần mềm đo hiệu năng truyền thông NetPIPE có thể được sử dụng để đánh giá hiệu năng của các thư viện truyền thông điệp theo chuẩn MPI. Phần mềm này cho phép định hướng biên dịch có sử dụng các hàm truyền thông điệp hoặc không, so sánh tốc độ truyền thông mạng trong trường hợp có và không sử dụng thư viện MPI sẽ cho thấy hiệu năng của thư viện này

9.3 Quy trình đánh giá hiệu năng hệ thống tính toán song song ghép cụm

Nội dung phần này sẽ trình bày về việc xây dựng quy trình quy trình đánh giá hiệu năng của một hệ thống tính toán song song ghép cụm dựa trên các phần mềm đánh giá hiệu năng mã nguồn mở thông dụng trên Thế Giới.

9.3.1 Xây dựng quy trình đánh giá hiệu năng của hệ thống tính toán song song ghép cụm

Qua định nghĩa cũng như kiến trúc, có thể thấy những yếu tố ảnh hưởng đến năng lực tính toán của hệ thống tính toán song song ghép cụm là:

- Tốc độ vi xử lý trên từng nút tính toán,
- Dung lượng bộ nhớ trong của từng nút tính toán,
- Tốc độ truyền thông của mạng LAN,
- Các phần mềm thiết lập môi trường tính toán.

Từ việc xác định các yếu tố ảnh hưởng đến năng lực tính toán của toàn bộ hệ thống, cùng với việc khảo sát, phân tích chức năng các phần mềm đánh giá hiệu năng chuyên dụng trên Thế Giới, nhóm nghiên cứu đã đề xuất một quy trình đánh giá hiệu năng của hệ thống tính toán song song ghép cụm như sau:

Tên phần mềm	Chức năng	Mục đích
Phần mềm đo hiệu năng tính toán **High Performance Linpack (HPL)** biên dịch chạy trên một nút	Đánh giá hiệu năng tính toán CPU đơn trên một nút tính toán	Đưa ra tốc độ tính toán cao nhất của từng máy trạm riêng lẻ
Phần mềm đo hiệu năng truy cập bộ nhớ trong	Đánh giá hiệu năng truy cập bộ nhớ trong trên một nút tính toán	Xác định kích thước bài toán thích hợp nhất đối với từng nút tính toán, từ đó có các cách

Stream		song song hóa và đệ trình công việc thích hợp đối với các ứng dụng song song cụ thể
Phần mềm đo hiệu năng truyền thông **Netperf**	Đánh giá hiệu năng truyền thông trong mạng LAN giữa hai nút tính toán bất kỳ theo giao thức TCP/IP	Xác định kích thước gói tin là bao nhiêu thì tốc độ truyền thông là cao nhất giữa hai nút tính toán, đồng thời đánh giá được khả năng truyền thông giữa các nút tính toán trong hệ thống
Phần mềm đo hiệu năng truyền thông **NetPipe**	Đánh giá hiệu năng truyền thông trong mạng LAN giữa hai nút tính toán bất kỳ thông qua các hàm gửi và nhận thông điệp của thư viện LAM/MPI	• Xác định kích thước gói tin là bao nhiêu thì tốc độ gửi và nhận thông điệp giữa hai nút tính toán thông qua môi trường tính toán song song thiết lập bởi LAM/MPI là cao nhất. • Sự so sánh kết quả giữa hai phần mềm đo hiệu năng Netperf và NetPipe cho người quản trị biết được các thông số cấu hình khi thiết lập môi trường tính toán song song với MPI đã thật sự tối ưu chưa.
Phần mềm đo hiệu năng tính toán **High Performance Linpack (HPL)** biên dịch chạy trên nhiều nút	Đánh giá hiệu năng tính toán của một số nút tính toán và của toàn bộ hệ thống	• Khảo sát sự thay đổi hiệu năng tính toán của hệ thống với số nút tham gia tính toán khác nhau • Xác định hiệu năng tính toán cao nhất của toàn bộ hệ thống. Kết quả này có thể sử dụng để so sánh hiệu năng của các

		hệ thống tính toán khác nhau.

Bảng 9-2 Các phần mềm sử dụng để xây dựng bộ công cụ đánh giá hiệu năng

9.3.2 Kết quả đánh giá hiệu năng hệ thống BKluster

9.3.2.1 Đánh giá hiệu năng tính toán

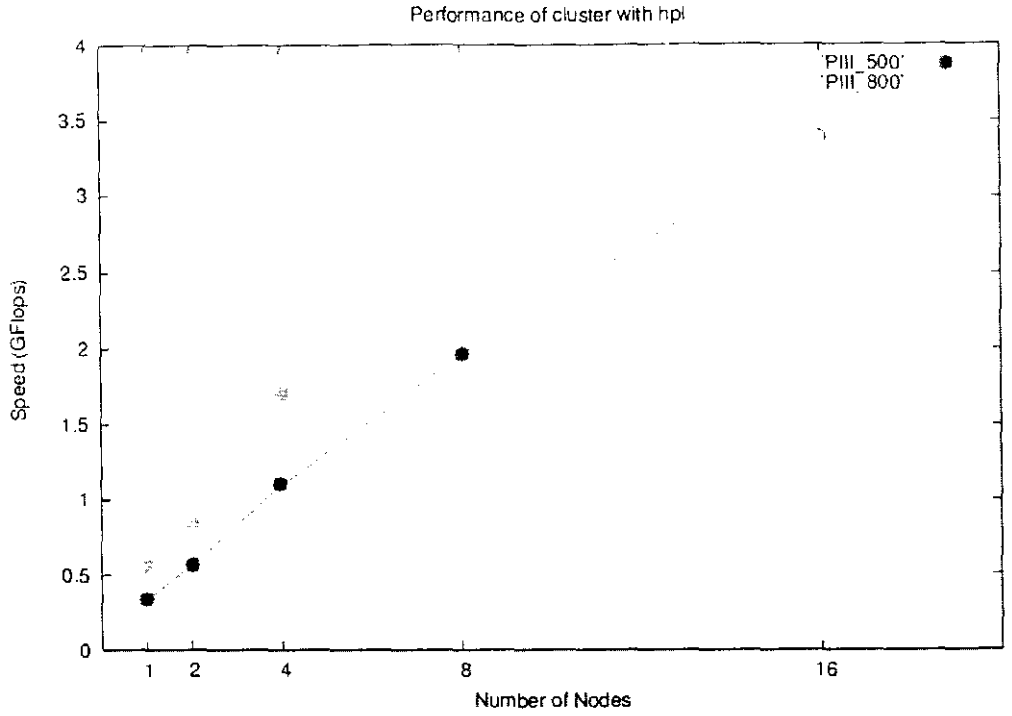
BKluster được đánh giá hiệu năng tính toán với số nút tính toán khảo sát lần lượt là 1, 2, 4, 8 và 16. Do hệ thống bao gồm 8 nút tính toán PentiumIII 500 MHz và 8 nút tính toán PentiumIII 800 MHz nên quá trình đo hiệu năng tính toán sẽ được thực hiện trên 1, 2, 4 và 8 nút cùng loại, sau đó là toàn bộ 16 nút của hệ thống

Số nút tính toán	PentiumIII 500 MHz	PentiumIII 800 MHz
1	0.334	0.544
2	0.562	0.857
4	1.094	1.703
8	1.965	3.066
16	3.396	

Bảng 9-3 Kết quả khảo sát hiệu năng (RMAX) của hệ thống BKluster (đơn vị là GFLOPS)

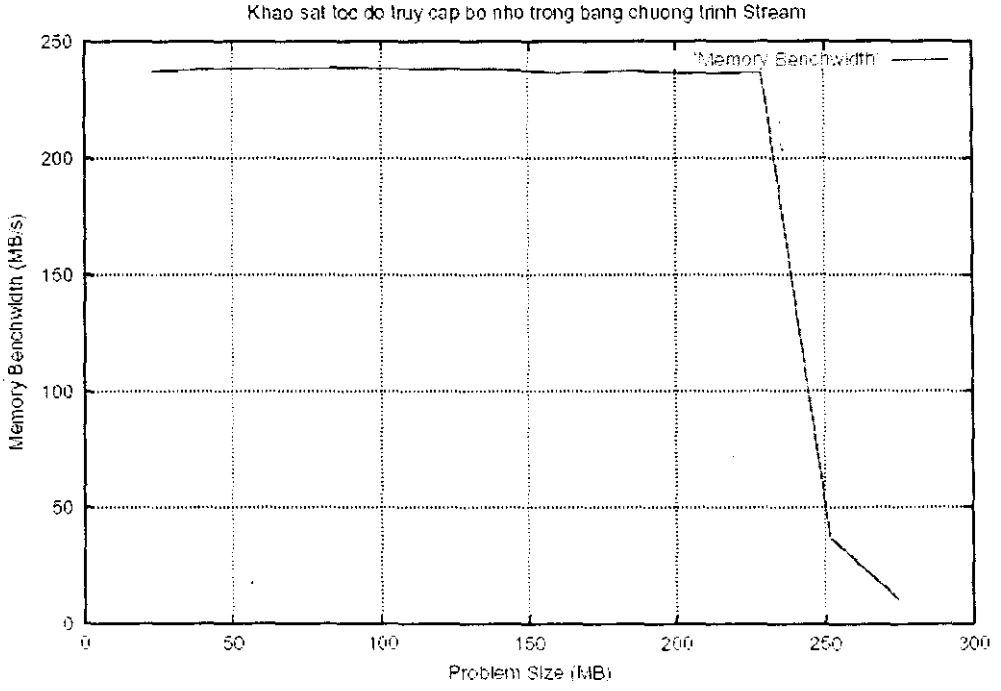
Với 16 nút tính toán, ta có *RPEAK = 8*0.5 + 8*0.8 = 10.4 (GFLOPS)*

Tỷ lệ RMAX/RPEAK = 3.396/10.4 = 32.7 %



Hình 9-1 Kết quả đánh giá hiệu năng tính toán hệ thống BKluster

9.3.2.2 Đánh giá hiệu năng truy cập bộ nhớ trong



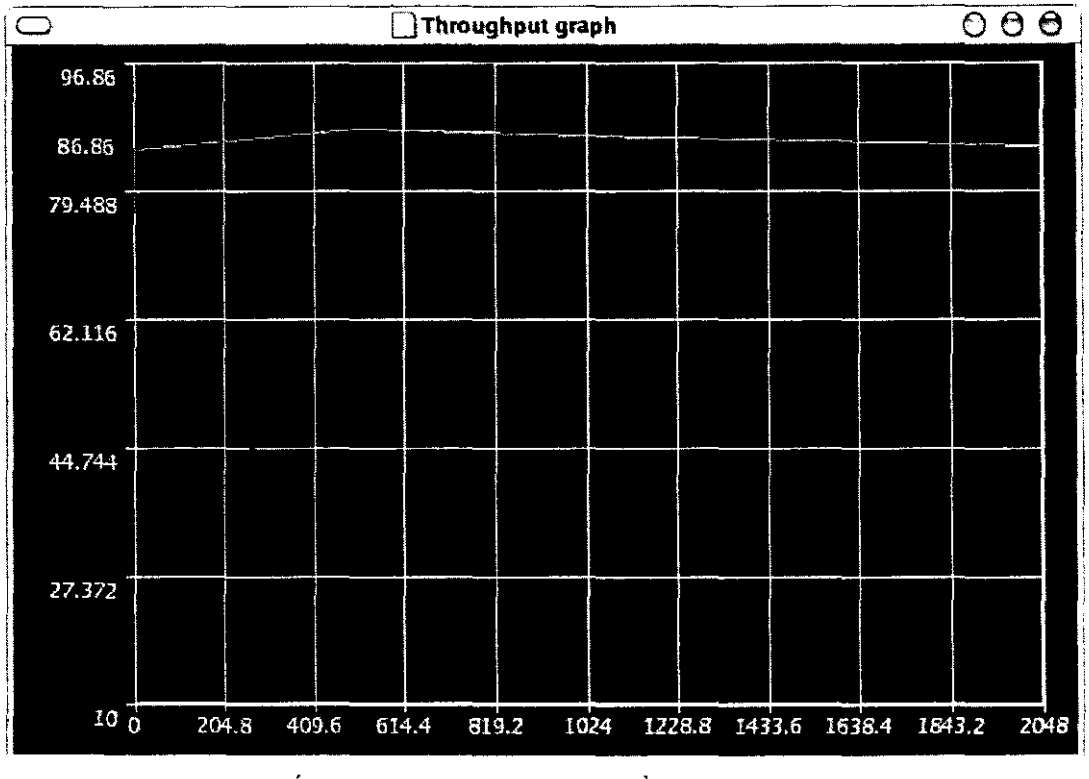
Hình 9-2 Kết quả đánh giá hiệu năng truy cập bộ nhớ trong của nút tính toán

Kết quả đo hiệu năng truy cập bộ nhớ trong với phần mềm Stream cho thấy: Trước khi xảy ra hiện tượng truy cập vào bộ nhớ swap (khi tổng giá trị các , tốc độ truy cập bộ nhớ trong luôn ổn định và bằng 236 – 238 MB/s. Khi xảy ra hiện tượng truy cập swap, tốc độ truy cập bộ nhớ trong sẽ giảm xuống đột ngột và sẽ giảm rất nhanh nếu kích thước của bài toán tiếp tục tăng lên

Kết quả trên cho thấy khả năng tính toán của máy trạm là ổn định với những bài toán có tổng giá trị các biến cần cấp phát nhỏ hơn 240 MB.

9.3.2.3 Đánh giá hiệu năng truyền thông theo giao thức TCP/IP

Thực hiện đo hiệu năng truyền thông theo giao thức TCP/IP bằng phần mềm NetPerf giữa hai nút tính toán trong mạng LAN, kết quả cho thấy tốc độ truyền thông giữa hai nút tính toán trong mạng là ổn định và dao động không nhiều trong khoảng lân cận 85 Mbps.

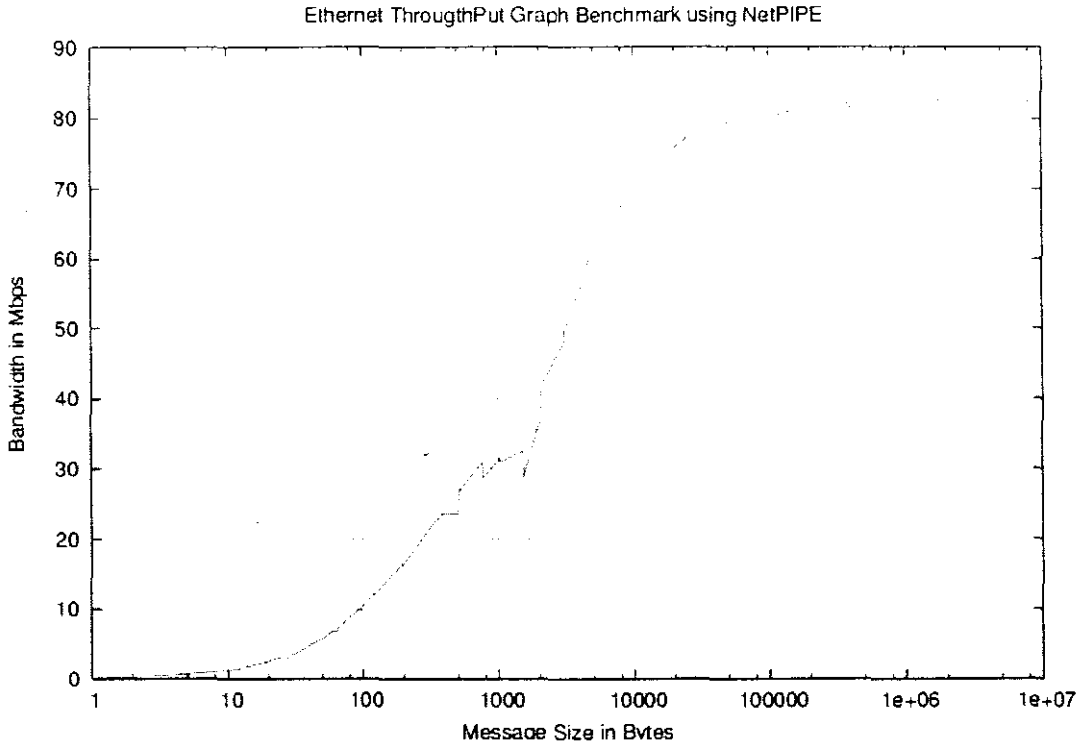


Hình 9-3 Kết quả đánh giá hiệu năng truyền thông theo giao thức TCP/IP

9.3.2.4 Đánh giá hiệu năng truyền thông môi trường song song LAM/MPI

Quá trình đánh giá hiệu năng truyền thông dựa trên các LAM daemon cho thấy tốc độ truyền thông trong mạng đạt được cao nhất là khoảng 82.82 Mbps với các gói

tin lân cận 6.2 MB. Điều này cho phép khẳng định rằng, môi trường tính toán song song đã được thiết lập tốt, không ảnh hưởng nhiều đến tốc độ truyền thông giữa các nút tính toán.



Hình 9-4 Kết quả đánh giá hiệu năng truyền thông của môi trường song song LAM/MPI

9.4 Bộ công cụ đánh giá hiệu năng hệ thống phân cụm

Các phần mềm đo hiệu năng được cung cấp từ nhiều nguồn khác nhau, viết bằng nhiều ngôn ngữ lập trình như C, C++, Fortran. Có những chương trình chỉ được viết bằng duy nhất một ngôn ngữ như NetPerf, NetPIPE viết bằng ngôn ngữ C, Linpack viết bằng Fortran, nhưng cũng tồn tại hiện tượng trong một gói phần mềm có các chương trình con được viết bằng nhiều ngôn ngữ khác nhau như các chương trình đo hiệu năng NPB. Hơn nữa, mỗi chương trình sẽ có thể liên kết đến nhiều thư viện khác nhau, trong đó có thư viện sẵn có trong hệ điều hành như pthread.h, math.h,... nhưng đồng thời cũng có những thư viện chuyên dụng như LAM/MPI, MPICH.

Các phần mềm thường được cung cấp mã nguồn mở dưới dạng một tập các file

nguồn, file header, không phải phần mềm nào cũng được cung cấp kèm theo hai file trợ giúp biên dịch là config và makefile. Vì vậy, khi dịch và chạy chương trình đôi khi người thực hiện cần phải trực tiếp đưa vào những tham số hoặc thay đổi một vài tham số trong makefile cho phù hợp với hệ thống cụ thể.

Vì những lý do trên mà ta cần đến một công cụ tương đối trong suốt hơn về mặt hệ thống để quá trình đánh giá hiệu năng có thể tiến hành một cách thuận lợi hơn. Công cụ đánh giá hiệu năng hệ thống sẽ tự động hoá việc thực hiện các công việc sau :

- Thực hiện các chương trình đánh giá hiệu năng với các giá trị đầu vào thay đổi. Việc thay đổi các giá trị đầu vào có thể do người sử dụng thực hiện hoặc được xác định một cách tự động dựa trên các tham số thực tế của hệ thống BKluster.
- Vẽ biểu đồ dựa trên một số giá trị đầu ra: quá trình vẽ biểu đồ được thực hiện ngay trên giao diện đồ họa hoặc sử dụng các phần mềm vẽ biểu đồ chuyên dụng như GnuPlot. Do đó, chương trình có khả năng lưu kết quả ra file theo định dạng tương thích với GnuPlot.

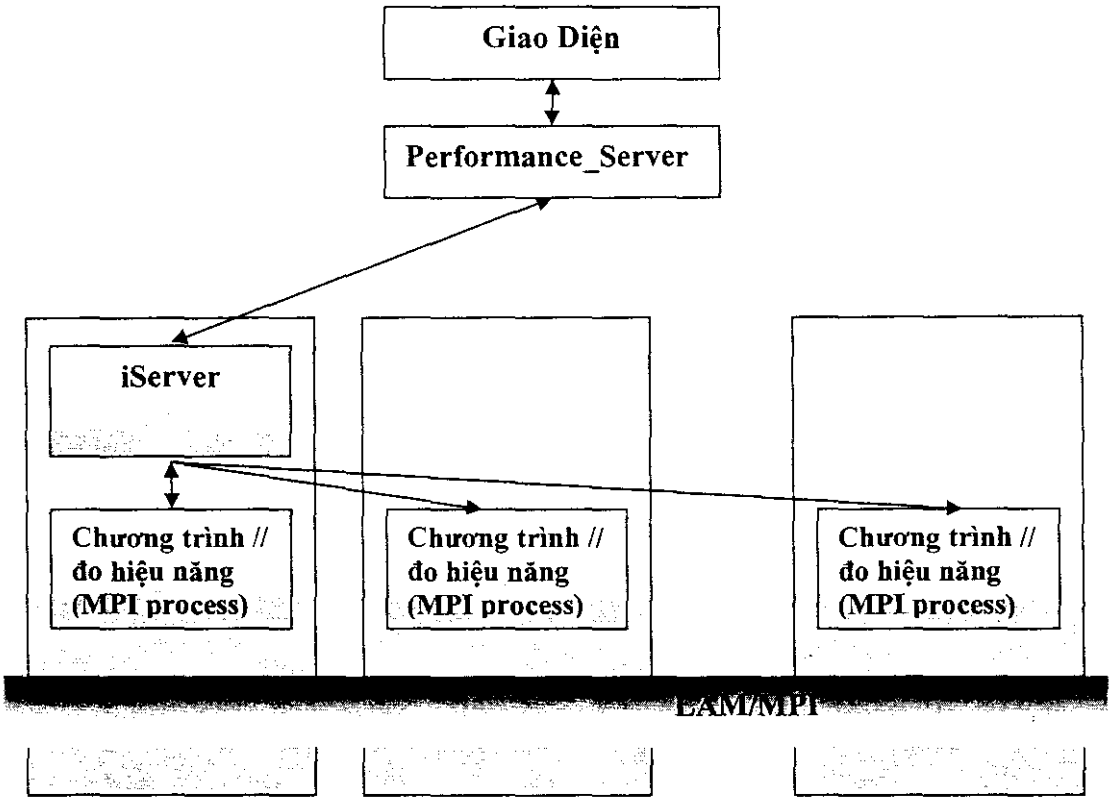
9.4.1 Công cụ đánh giá hiệu năng tính toán của hệ thống

Công cụ đánh giá năng lực tính toán của hệ thống cho phép biên dịch và chạy một số chương trình đánh giá hiệu năng thông dụng trên Thế Giới như: xHPL, NPB, NETPIPE, ... trên một số nút tính toán xác định. Các chương trình đánh giá hiệu năng tính toán đều có dạng một chương trình song song, thực hiện một số lượng lớn phép toán số thực dấu phẩy động, có khả năng biên dịch theo một số chuẩn thông dụng, đặc biệt là chuẩn MPI.

Quá trình đánh giá hiệu năng tính toán được thực hiện qua những bước như sau:

- Người quản trị thông qua thành phần giao diện xác định các thông số như: tên phần mềm đo hiệu năng, tổng số nút trong cụm cần đo hiệu năng, ...
- Performance_Server gửi các thông số trên đến iServer trên một nút bất kỳ trông số tổng các nút cần đánh giá hiệu năng
- iServer thực hiện các việc sau:

- Biên dịch chương trình đánh giá hiệu năng (Source code và file thực thi sau khi biên dịch của các chương trình đánh giá hiệu năng được đặt trên bộ nhớ dùng chung của BKluster)
- Chạy chương trình đánh giá hiệu năng dưới dạng một chương trình song song viết theo chuẩn MPI trong môi trường LAM
- Gửi kết quả về Performance_Server



Hình 9-5 Kiến trúc thành phần đánh giá hiệu năng tính toán hệ thống

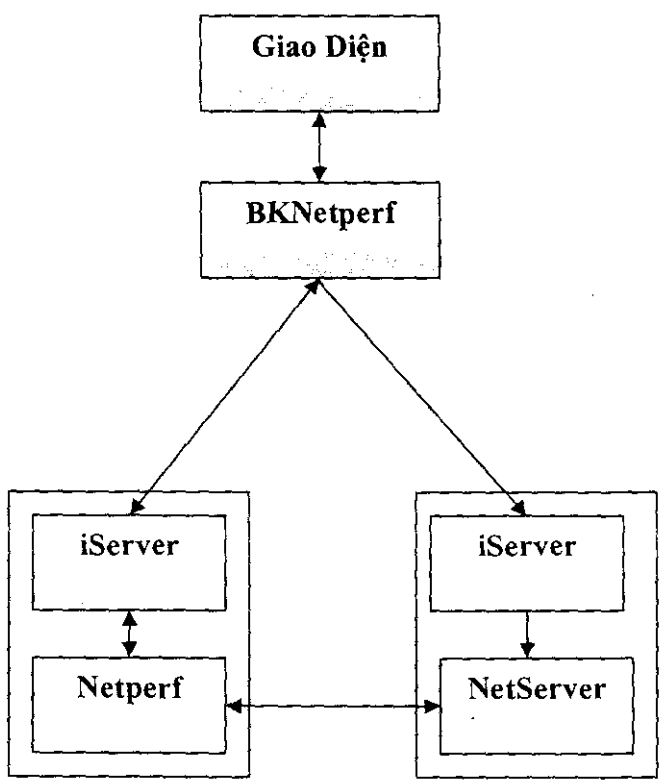
9.4.2 Công cụ đánh giá hiệu năng truyền thông trong hệ thống

Công cụ đánh giá hiệu năng truyền thông hệ thống được phát triển dựa trên phần mềm mã nguồn mở Netperf. Netperf bao gồm 2 module Netserver và Netperf chạy trên 2 máy tính khác nhau. Module Netperf sẽ gửi các gói tin đến module Netserver, dựa vào kết quả phản hồi của Netserver sẽ tính được tốc độ truyền thông trong mạng nối giữa hai máy tính.

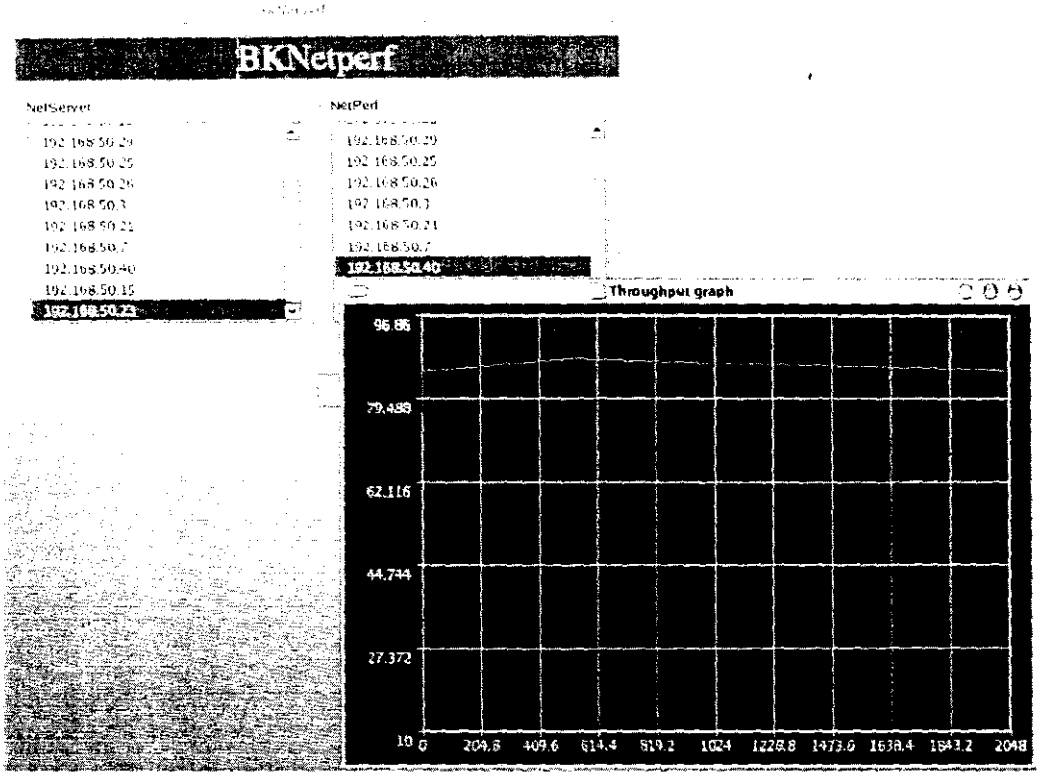
Cấu trúc công cụ đánh giá năng lực truyền thông được mô tả trong hình vẽ sau. Các module Netperf, NetServer và iServer đã được cài đặt sẵn trên mọi node tính toán

Quá trình đánh giá năng lực truyền thông được thực hiện qua những bước như sau:

- Người quản trị thông qua thành phần giao diện xác định các thông số như: kiểu gói tin, kích thước gói tin, tên máy nguồn, tên máy đích, ...
- BKNetperf gửi các thông số trên đến iServer trên máy nguồn và máy đích,
- iServer trên máy đích khởi động chương trình NetServer
- iServer trên máy nguồn gọi chương trình Netperf, thực hiện truyền tin theo các thông số đã xác định
- iServer trên máy nguồn trả kết quả về cho BKNetperf



Hình 9-6 Kiến trúc công cụ đánh giá hiệu năng truyền thông hệ thống



Hình 9-7 Giao diện phần mềm BKNetPerf đánh giá hiệu năng truyền thông

Tài Liệu Tham Khảo

[1]. **Thomas Sterling**, "*An Introduction to PC Clusters for High Performance Computing*", *California Institute of Technology and NASA Jet Propulsion Laboratory, USA*

[2]. **Albeaus Bayucan Robert L. Henderson Casimir LesiakBhroam Mann Tom ProettDave Tweten**, "Portable Batch System External Reference Specification". *MRJ Technology Solutions*

[3]. **Albeaus Bayucan Robert L. Henderson Casimir LesiakBhroam Mann Tom ProettDave Tweten**, "Portable Batch System Internal Design Specification". *MRJ Technology Solutions*

[4]. **Albeaus Bayucan Robert L. Henderson Casimir LesiakBhroam Mann Tom ProettDave Tweten,** "Portable Batch System Administrator Guide". *MRJ Technology Solutions*

[5]. **Albeaus Bayucan Robert L. Henderson Casimir LesiakBhroam Mann Tom ProettDave Tweten,** "Portable Batch System user guide". *MRJ Technology Solutions*

[6]. **PACS Training Group**, "Introduction to MPI", *University of Illinois.*

[7]. LAM/MPI Forum, "www.lam-mpi.org ".

[8]. **The LAM/MPI Team**, "LAM/MPI Documents", *Open Source Lab*

[9]. **Nicolas J. Nevin**, "The XMPI API and trace file format", *January 29, 1997.*

[10]. "UC Berkeley Grid" *http://www.millennium.berkeley.edu/ganglia/*

[11] Karl-Johan Andersson, Daniel Aronsson and Patrick Karlsson. "**An evaluation of the system performance of a Beowulf cluster**", *Internal Report No. 2001:4* .

[12] Roger Hockney. "**A Framework for Benchmark Performance Analysis**", *Supercomputer 48, IX-2, 1992*

[13] Atul Kumar. "**A Performance Evaluation and Benchmarking Tool,**

Department of Computer Science & Engineering", *Indian Institute of Technology, Kanpur January 1998*

[14] D.Bailey, E.Barszcz, L. Dagum and H. Simon. "**NAS Parallel Benchmark Result**", *RNR-94-006, March 211994*

[15] Jack J. Dongarra, Piotr Lszczek, Antoine Petitet "**The Linpack Benchmark : Past, Present, and Future**", *December 2001*

[16] John D. McCalpin. "**STREAM : Sustainable Memory Bandwidth in High Performance Computers**", *http://www.cs.virginia.edu/stream/*

[17] Quinn O.Snell, Armin R. Mikler and John L. Gustafson. "**NetPIPE : A Network Protocol Independent Performance Evaluator**", *Ames Laborator Scalable Computing Lab, Ames, Iowa 50011, USA*

[18] "Netperf : A Network Performance Benchmark, Revision 2.1". *Informtion Networks Divison Hewlett-Packard Company February 15, 1996*

[19] Ray Flanery, Al Geist, Brian Luethke, Jens Schwidder, et Stephen Scoţt "**The Scalable System Administrator: via C3 & M3C Tools**" *Computer Science & Mathematics Division Oak Ridge National Laboratory Oak Ridge, TN 37830-6367 USA*

[20] Putchong Uthayopas - Sompong Techarphonchai, Tawee Iam-ngamsup Parallel Research Group "**VISUALIZING A LARGE SCALE BEOWULF CLUSTER USING WEB AND VRML TECHNOLOGY**", *CONSYL Department of Computer Engineering, Faculty of Engineering, Kasetsart University, Bangkok, Thailand*

[21] Janet Ellsworth, Patrick Meehan, Vallard Benincosa, Bruce Potter "**Using IBM Cluster Systems Management (CSM) for Linux in xCAT Environments - White Paper**"

[22] Christine Morin "**Gestion des ressources pour le calcul haute performance dans les grappes : état des lieux et perspectives**" - *IRISA/Université de Rennes 1*

[23] Renaud Lottiaux et Christine Morin IRISA/INRIA "**A Cluster Operating System Based on Software COMA Memory Management**", *Campus de Beaulieu 35042 Rennes Cedex, France*

[24] Rajkumar Buyya, "**SINGLE SYSTEM IMAGE (SSI)**" - *MONASH*

UNIVERSITY, AUSTRALIA, Toni Cortes - UNIVERSITAT POLITECNICA DE CATALUNYA, SPAIN, Hai Jin - HUAZHONG UNIVERSITY OF SCIENCE AND TECHNOLOGY, CHINA

[25] Barak, A., et La'adan "**The MOSIX multicomputer operating system for high performance cluster computing (ftp)**" *(Journal of Future Generation Computer Systems - Available: http://www.mosix.cs.huji.ac.il/)*

[26] http://www.adelielinux.ca

[27] Dr. Bruce J. Walker, "**Using OpenSSI Linux Clusters in all forms of Cluster Environments OpenSSI.org**" - *HP Fellow Office of Strategy and Technology*

[28] "**Qt 3.3 Whitepaper**" *http://www.trolltech.com*

[29] "**Réseaux d'interconnexion dans les grappes de calcul Brice Goglin**" *http://perso.ens-lyon.fr/brice.goglin/teaching*

[30] "Grid3" http://gocmon.uits.iupui.edu/ganglia-webfrontend/

[31] Espen S. Johnsen, Otto J. Anshus, John Markus Bjørndalen, Lars Ailo Bongo, "**Survey of execution monitoring tools for computer clusters**" *September 29, 2003 - Department of Computer Science, University of Tromsø.*

[32] *"XML" http://www.w3schools.com/*

[33] *"SSI-OSCAR" http://ssi-oscar.irisa.fr/*