Системы управления кластерами

А.И. Аветисян, Д. А. Грушин, А.Г. Рыжов

1. Введение

В данном обзоре мы рассмотрим одну из параллельных архитектур, называемую кластерной архитектурой. Согласно [10], кластер – это группа вычислительных машин (называемых узлами кластера), объединенных коммуникационной средой и работающих как единое целое. Каждый узел кластера работает под управлением своей операционной системы. Однородным кластером называется кластер, состоящий из узлов, имеющих одинаковую конфигурацию, и работающих под управлением одинаковых операционных систем. Далее мы будем рассматривать только однородные кластеры.

В середине 60-х годов идея кластера была предложена компанией IBM как способ объединения мейнфреймов для построения относительно недорогого суперкомпьютера. Более широкое распространение кластерные архитектуры получили позднее, в 80-е годы, с появлением быстрых микропроцессоров, достаточно производительных каналов связи и средств написания параллельных приложений. В настоящее время конструирование кластеров сдвигается в сторону построения систем на основе стандартных компонент: обычных двух- или четырехпроцессорных рабочих станций (или персональных компьютеров) и коммуникационного оборудования (Myrinet, SCI, Fast Ethernet и др.).

Кластеры применяются для решения различных классов задач: они используются и для выполнения большого массива последовательных программ, никак не связанных одна с другой, и для обработки данных *распределенными* программами, состоящими из большого числа независимых процессов, имеющих общие данные (coarse-grain parallelism), и для выполнения *параллельных* программ, когда выполнение отдельных операторов программы распределяется по узлам вычислительной сети, чтобы сократить время выполнения программы (fine-grain parallelism). Каждый из вышеперечисленных классов задач предъявляет свои требования к свойствам и параметрам кластера. Эти требования подробно изложены в [10]. Основное требование к кластеру и его системе управления состоит в том, что для классов программ, которые предполагается выполнять на кластере, он должен быть

масштабируем – при увеличении числа узлов кластера должна пропорционально возрастать его производительность.

В простейшем случае, когда на кластере требуется выполнить только одну параллельную программу, вся функциональность, необходимая для запуска, выполнения и завершения программы, может обеспечиваться операционными системами на узлах и самой программой. В более сложных случаях, когда требуется выполнить несколько задач, как параллельных, так и последовательных, имеющих различные требования к системным ресурсам (например, параллельные программы, занимающие только часть узлов кластера), в дополнение к среде выполнения (т.е. операционной системе узла и, возможно, дополнительных библиотек, поддерживающих выполнение параллельных программ), необходима некоторая система, управляющая вычислительными ресурсами кластера в целом.

В [17] кластер определяется как множество рабочих станций (узлов), связанных коммуникационной средой и способных работать как единое целое за счет дополнительного программного обеспечения, называемого системой управления кластером. Система управления кластером в очень ограниченном смысле может рассматриваться как распределенная операционная система для кластера. Она обеспечивает управление над выполняемыми заданиями, пользователями и ресурсами. Когда кластер применяется для выполнения большого числа последовательных программ, масштабируемость кластера во многом определяется его системой управления. Масштабируемость кластера, применяемого для выполнения распределенных программ, зависит от свойств аппаратуры, однако, если требуется одновременно выполнить несколько распределенных программ, каждая из которых занимает часть узлов кластера, масштабируемость снова начинает зависеть от его системы управления.

Система управления должна распределять ресурсы кластера между поступающими от пользователей заданиями. Для этого система может иметь очередь или набор очередей, распределять (выделять необходимое количество узлов для запуска) каждого задания таким образом, чтобы максимально использовать вычислительные возможности кластера и, наконец, собирать полученные результаты. Более подробный список основных функций, обеспечиваемых системами управления, был впервые сформулирован в работе [17] и, позднее, в [5].

К настоящему времени разработано достаточно много систем управления. В данном обзоре мы постараемся рассмотреть некоторые наиболее интересные из существующих систем.

В разделе **2** мы рассмотрим основные функции, обеспечиваемые системой управления кластером, в разделе **3** будут рассмотрены основные архитектурные особенности систем управления, в разделе **4** коротко опишем наиболее интересные системы управления кластером. В разделе **5** на основе

анализа систем, рассмотренных в разделе **4**, сформулируем требования к системе управления кластером, разрабатываемой в ИСП РАН.

2. Основные функции, обеспечиваемые системой управления кластером

2.1. Классы задач

Во введении мы упомянули, что кластеры применяются для выполнения следующих классов программ:

- массив последовательных программ, никак не связанных одна с другой;
- распределенные программы, состоящие из независимых процессов, имеющих общие данные (coarse-grain parallelism);
- параллельные программы, (fine-grain parallelism);

Распределенные программы – это основной тип задач, для которых создаются кластеры. Каждая такая программа состоит из конечного множества процессов, обменивающихся данными через коммуникационную среду кластера. В большинстве случаев во время выполнения таких программ часть узлов может простаивать.

Рассмотрим пример. Предположим, в некоторый момент времени в очереди находится несколько распределенных программ готовых к запуску, и кластер имеет M свободных узлов. Пусть каждая такая программа требует для своей работы некоторое количество узлов K: $1<K<N$, где N – общее число узлов кластера. Если

$$M < K \qquad (1),$$

то программу запустить нельзя, и узлы в таком случае будут простаивать. Отсюда видно, что чем меньше узлов K требует программа, чем меньшее количество узлов M будет простаивать.

В такой ситуации на помощь приходят последовательные программы. Каждая такая программа выполняется независимо и требует всего один узел. Более того, последовательные программы не используют вычислительную коммуникационную среду, не создавая тем самым дополнительной нагрузки. В большинстве случаев последовательные программы объединяются в группу. Группа может состоять из различных программ, имеющих общие начальные данные или из одной программы, каждый запуск которой производится с различными входными данными.

Типичным примером может служить поиск генов в базе данных. Программа поиска запускается с каждым новым описанием гена и ищет похожие гены в

одной и той же базе данных. Каждый поиск выполняется независимо от других, тем самым, требуя только один узел. С другой стороны, чем больше свободных узлов, тем быстрее будет произведен поиск набора генов.

В общем случае последовательные программы можно рассматривать как параллельные, требующие всего один узел. Если вернуться к формуле (1), то можно сказать, что, имея в очереди некоторое достаточное количество последовательных программ, можно полностью избавиться от простаивающих узлов.

При запуске программы или группы программ пользователь должен указать некоторые параметры, например количество необходимых процессоров или узлов, количество памяти. Помимо этого каждая программа может использовать для своей работы некоторые файлы, например база данных или файл с описанием гена из предыдущего примера. Эти файлы должны быть переданы на конкретный узел до того, как программа будет запущена. Указывать каждый раз всю эту информацию при запуске задачи потребует много времени и может стать причиной множества неудобств и ошибок. Обрабатываемые системой задачи должны описываться заданием. Для системы управления оно определяет действия, которые необходимо совершить при запуске и выполнении задачи или группы задач, и при сборе результатов вычислений. Задание обычно представляется в виде отдельного текстового файла, содержащего описание необходимых действий и который затем интерпретируется системой управления. Файл с описанием задания, исполняемый файл задачи, а также все необходимые для работы файлы данных образуют пакет или пакетное задание.

В процессе выполнения программа может создавать другие файлы, некоторые из которых содержат результат работы и должны быть возвращены пользователю. Более того, некоторые файлы могут являться входными данными для других задач, тем самым, создавая зависимость по результатам. Следует упомянуть, что система должна иметь возможность описывать такие зависимости в файле задания. Помимо создаваемых задачей файлов весь производимый в процессе выполнения стандартный вывод (печать на экран), а также стандартный поток сообщений об ошибках перенаправляются в специальные выходные файлы. Пользователь может также задать системе файл стандартного ввода, в таком случае система должна иметь возможность имитировать диалог с выполняющейся программой, аналогично возможностям программы expect[1] [12]. Все указанные файлы создаются в рабочем каталоге задачи. Этот каталог может находиться на удаленном файловом сервере или непосредственно на узле, в таком случае система должна самостоятельно копировать все необходимые файлы. Следует отметить следующее. Если

[1] Программа *expect* позволяет автоматизировать диалог с интерактивными программами, такими как telnet, ftp, passwd, fsck, rlogin, tip и многими другими.

система управления использует только сетевую файловую систему, то при большом количестве узлов (большие кластеры могут состоять из нескольких сотен узлов) и операций ввода-вывода может возникнуть большая загрузка передающей среды (обычно Ethernet) и работа системы может практически остановиться.

Система управления кластером может позволять выполнять интерактивные задания. Ввод, вывод и сообщения об ошибках в таком случае перенаправляются на машину пользователя. Интерактивный режим в большинстве случаев необходим для отладки или тестирования программ, когда требуется постоянный анализ пользователем получаемых данных или когда описание имитации диалога объемно и требует слишком много времени. Сложность выполнения интерактивных задач заключается в необходимости поддерживать терминальное соединение с узлом на протяжении всего времени работы. Существуют различные способы, позволяющие сократить продолжительность соединения, например открытие по запросу и закрытие при отсутствии активности.

После того, как пользователь отправляет любое задание на выполнение, он уже не может полностью им управлять так, как это возможно на обычной рабочей станции. Система управления должна предоставлять пользователю возможность проверки статуса выполняемой или выполненной задачи, т.е. завершена программа или еще выполняется, какое количество памяти, процессорного времени использует программа. Система может предоставить эту информацию пользователю, используя электронную почту, SMS сообщения или диалоговую программу. Пользователь должен иметь возможность непосредственно управлять ходом выполнения отправленной программы, например, остановить программу или запустить заново.

Рассмотрим выполнение системой параллельных программ. Согласно [10], поддержку параллельных программ можно разделить на следующие критерии:

1. Система имеет фиксированную встроенную поддержку для одной или нескольких параллельных программных сред, например PVM или MPI. Такой подход требует перекомпоновки программы со специальной библиотекой.
2. Система предоставляет возможность выбора узлов для параллельной программы. Например, многие параллельные программные среды задают список конкретных узлов для запуска параллельной программы в командной строке, система может изменять этот список после того, как подходящие узлы будут выбраны. Основная проблема состоит в том, что порождение процессов происходит внутри параллельной программы и система не в состоянии отследить такие процессы.
3. Система управления и параллельная программная среда могут общаться, использую некий заранее определенный протокол. Такой подход был предложен инициативной группой psched [31]. Программный интерфейс psched должен поддерживаться как системой управления, так и

параллельной средой. В настоящее время наиболее популярные среды MPI и PVM не поддерживают интерфейс psched.

Кластеры также могут использоваться для выполнения другого типа параллельных программ – SPMD, когда выполнение отдельных операторов программы распределяется по узлам кластера [4].

2.2. Балансировка нагрузки

Система управления может отправлять задачи на выполнение на конкретные узлы в кластере не просто по мере их получения от пользователей, а, таким образом, чтобы каждый узел, входящий в состав кластера, выполнял работу пропорционально своей мощности и не простаивал. Балансировка может быть статической, до запуска задачи, и динамической, уже во время работы задачи. Статическая балансировка привлекательна тем, что не требует дополнительных затрат во время выполнения задачи, однако динамическая может быть более эффективна в случае изменения физической структуры кластера, добавления или удаления узлов во время работы. Рассмотрим оба эти способа.

Для того чтобы эффективно производить статическую балансировку нагрузки, необходимо иметь некоторую информацию о каждой задаче, например, необходимое количество узлов, оперативной и дисковой памяти, предельное время работы. Вся эта информация является списком ресурсов, необходимых задаче, и описывается в файле задания. Используя указанные ресурсы, система способна выбирать из очереди готовых к выполнению задач наиболее подходящие в конкретной ситуации. Существует множество решений задачи статического распределения, которая для большинства вариантов принадлежит классу NP-полных задач, за исключением некоторых очень простых случаев [13, 32]. В целом следует сказать, что наиболее логичным способом распределения задач является минимизация количества простаивающих машин, в таком случае кластер используется наиболее эффективно.

Динамическая балансировка позволяет перераспределять уже запущенные задачи. В таком случае используется миграция процессов. Миграция процессов позволяет перемещать процесс с одной машины на другую без необходимости перезапуска процесса с самого начала. Миграция процессов привлекательна тем, что поступление задач и ресурсов от пользователей не всегда известно заранее. Миграция в таком случае дает системе еще один шанс правильно распределить выполняемые задачи [3]. Миграция обычно используется вместе с контрольными точками, образ процесса сохраняется на одной машине и затем восстанавливается на другой. Более подробно мы расскажем о контрольных точках в разделе 2.3 и в разделе 4.2 при описании системы Condor.

Если система разрешает пользователю для своей задачи указать используемые ресурсы, то нет никаких гарантий, что после запуска задачи эти данные будут

соответствовать действительности. Система должна контролировать соответствие между реальным использованием ресурсов и теми данными, которые указал для данной задачи пользователь, и в случае несоответствия завершить задачу или произвести заранее предусмотренное для такой ситуации действие. По нашему мнению, чем больше будет такой контроль, тем больше гарантий безопасности для системы и ее пользователей. Возможность ограничения ресурсов определяется операционной системой, использующейся на узлах кластера, и архитектурой самой системы управления.

2.3. Защита от сбоев

Как мы сказали выше – система управления может рассматриваться как распределенная операционная система для кластера. Основным принципом любой распределенной системы является иллюзия целостности, когда совокупность независимых компьютеров представляется пользователю единой вычислительной машиной. Пользователь кластера отправляет задания на известный сетевой адрес. Этот адрес однозначно определяет некоторый узел в составе кластера, называемый управляющим узлом. Кластеры очень часто создаются с выделенным управляющим узлом, на котором не производятся вычисления, а работает основная часть системы управления: подсистема очередей, система сбора и обработки различной информации. Управляющий узел также может включать в себя файловый сервер и служить шлюзом, ограничивающим узлы кластера от внешней сети.

Во время работы кластера один из узлов может отказать. В таком случае все вычисления, производимые на данном узле, теряются. Система должна, так или иначе, реагировать на такие отказы аппаратуры и, в случае необходимости перезапускать остановленные задачи.

В простейшем случае каждая программа может быть запущена заново. Для программ, которые выполняются долгое время, перезапуск означает потерю большого количества вычислений и, следовательно, времени. Для сохранения текущего состояния программы, и затем ее восстановления используется механизм контрольных точек. В случае сбоя теряются только те вычисления, которые были произведены с момента последней контрольной точки. Контрольная точка процесса представляет собой образ адресного пространства процесса.

Существует два метода создания контрольных точек: последовательное и неблокирующее сохранение. Последовательное сохранение сокращает объем необходимых для сохранения данных. Сохраняются только те страницы памяти, которые были изменены с момента последнего сохранения. Неблокирующее сохранение позволяет процессу продолжать свое выполнение во время сохранения. Страницы памяти процесса блокируются на запись, если же процесс пытается изменить страницу, она копируется, и процесс сохранения продолжается уже с копией страницы. Сохранение образа процесса

может существенно затормозить работу машины, выполняющей такое сохранение. Помимо вычислительных ресурсов процесс создания контрольных точек требует дополнительного дискового пространства. Для систем управления процесс создания контрольных точек сложен и зависит от реализации системы управления и возможностей операционной системы, используемой на узлах кластера. Все системы, поддерживающие контрольное сохранение, делают это с теми или иными ограничениями, о которых мы упомянем при более детальном рассмотрении конкретных систем.

Сложной задачей является восстановление состояния параллельной программы. Применяются два метода (и их комбинация), основанные на промежуточной фиксации состояния либо ведении журнала выполняемых операций. Они различаются объемом запоминаемой информацией и временем, требуемым для восстановления. На рис. 1 показаны три процесса (X,Y,Z), взаимодействующие через сообщения. Вертикальные черточки показывают на временной оси моменты запоминания состояния процесса для восстановления в случае отказа. Стрелочки соответствуют сообщениям и показывают моменты их отправления и получения.

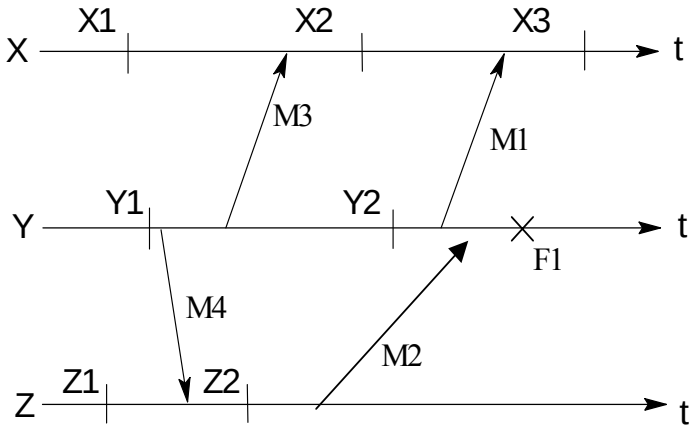

Рис. 1. *Иллюстрация эффекта домино*

Если процесс Y аварийно завершился в точке F1 после отправки сообщения M1, посылка которого зафиксирована в точке x3 и не зафиксирована в y2. Сообщение M1 называется сообщением-сиротой, а состояния y2 и x3 являются несогласованными. В таком случае процесс X должен быть возвращен в состояние x2. Этот эффект известен как эффект домино. Множество контрольных точек должно быть консистентным. Для любой зафиксированной операции приема сообщения, соответствующая операция посылки должна быть также зафиксирована (нет сообщений-сирот). Система должна

осуществлять восстановление с использованием консистентного множества контрольных точек – повторить отправку тех сообщений, квитанции о получении которых стали недействительными в результате отката.

В случае отказа управляющего узла, система должна иметь возможность либо продолжить работу, выбрав самостоятельно новый управляющий узел, либо восстановить свое состояние позднее, когда управляющая машина вернется в рабочее состояние.

2.4. Пользовательский интерфейс

Очень важная характеристика системы для конечных пользователей – это пользовательский интерфейс. Если пользователь не понимает систему из-за сложного или громоздкого интерфейса, система не будет использоваться.

Система управления должна предоставлять пользователю возможность доступа через WEB – интерфейс. Такой доступ не требует установки на машины пользователя никакого дополнительного программного обеспечения.

Задания для системы управления должны формулироваться с использованием некоторого языка. Этот язык не должен быть сложен при изучении и использовании.

Для обеспечения эффективности система может потребовать, чтобы программа была заново перекомпилирована или собрана с использованием специальных библиотек. Такое требование обычно необходимо для обеспечения поддержки контрольных точек и миграции процессов, однако в большинстве случаев может быть неприемлемо для пользователей. Переносимости в общем случае нет.

3. Особенности реализации систем управления кластерами

В общем случае система управления кластером имеет подсистему, работающую на управляющем узле кластера (управляющая подсистема) и подсистемы, работающие на каждом вычислительном узле (периферийные подсистемы). Управляющая и периферийные подсистемы взаимодействуют через коммуникационную систему кластера.

Следует отметить, что кластер должен иметь две коммуникационных системы: одна используется для взаимодействия компонент распределенных программ, выполняемых на кластере, другая – для организации взаимодействия подсистем системы управления кластером. Это следует из того, что для эффективного использования сетей рабочих станций коммуникационная система должна обеспечивать максимальную пропускную способность и минимальные задержки при передаче данных [22], и дополнительная нагрузка со стороны системы управления может значительно снизить производительность кластера. В настоящее время коммуникационная система,

обеспечивающая взаимодействие компонент распределенных программ, расположенных на разных узлах кластера, строится на основе коммуникационного оборудования Myrinet или SCI, а коммуникационная система для работы системы управления – на основе Ethernet или Fast Ethernet.

Управляющая подсистема принимает задания от пользователей и отправляет их на вычислительные узлы. Кроме того, управляющая подсистема опрашивает периферийные подсистемы узлов кластера и собирает необходимую информацию, связанную с текущим состоянием каждого узла. Все управление кластером осуществляется его управляющей подсистемой, а периферийные подсистемы узлов кластера в общем случае только отвечают на запросы управляющей подсистемы. Такой подход позволяет избежать резких скачков сетевого трафика, которые могут быть порождены несогласованными действиями большого количества периферийных подсистем. Можно сказать, что система управления кластером работает по принципу клиент-сервер, клиентом является управляющая подсистема, а каждая периферийная подсистема – сервером.

Как управляющая, так и периферийные подсистемы могут быть реализованы как прикладные программы для операционной системы узла. Однако некоторые требования к системе управления, связанные с механизмами защиты (сохранение состояния процесса, контроль над используемыми ресурсами и т.п.) не могут быть полностью реализованы на уровне прикладных программ. Для обеспечения указанных возможностей часть системы управления должна быть реализована на уровне ядра операционной системы (если операционной системой узла является Linux, то наиболее удобна реализация в виде модуля).

4. Анализ программных пакетов управления кластерами

В этом разделе предлагается сравнительный анализ наиболее распространенных систем управления кластером. Будут рассмотрены системы управления Beowulf, EnFuzion, MOSIX, Condor, LoadLeveler, LSF, PBS и Cleo. Будет также рассмотрен планировщик заданий MAUI. Соответствие систем вышеупомянутым требованиям можно выразить следующей таблицей (табл. 1). Рассмотрим эти системы более подробно.

	Beowulf	EnFuzion	MOSIX	Condor	Load Leveler	LSF	Cleo	PBS Pro
Пакетные задания	Да	Да	Да	Да	Да	Да	Да	Да
Файл задания	Да	Да	Нет	Да	Да	Да	Нет	Да
Зависимость задач по результатам	Нет	Нет	Нет	Нет	Нет	Да	Нет	Нет
Статус задачи	Да	Да	Да	Да	Да	Да	Да	Да
Балансировка	Нет	Нет	Да	Нет	Да	Да	Да	Да
Задание ресурсов	Да	Нет	Нет	Да	Да	Да	Да	Да
Задание приоритетов	Да	Да	Нет	Да	Да	Да	Да	Да
Интерактивные задачи	Нет	Нет	Да	Нет	Да	Да	Нет	Да
Ограничение ресурсов	Да	Нет	Нет	Нет	Да	Да	Нет	Да
Контрольные точки	Нет	Нет	Нет	Да	Да	Да	Нет	Нет
Миграция процессов	Нет	Нет	Да	Да	Да	Да	Нет	Нет
Перекомпиляция	Да	Нет	Нет	Да	Нет	Нет	Нет	Нет
Защита от сбоев	Нет	Да	Да	Да	Да	Да	Нет	Да
Параллельные программы	Да	Нет	Нет	Да	Да	Да	Да	Да
Монопольный режим	Да	Да	Да	Да	Да	Да	Да	Да
Интерфейс	Да	Нет	Да	Нет	Да	Да	Нет	Да

Таблица 1. *Сравнение систем управления кластерами*

4.1. Система Beowulf

В 1994 г. научные сотрудники CESDIS[2] Thomas Sterling и Don Becker собрали кластер, состоящий из 16 процессоров DX4, связанных сетью Ethernet [8]. Они назвали свой кластер Beowulf. Разработка быстро выросла в то, что сейчас называют проект Beowulf, который в настоящее время ведется компанией Scyld.

Система управления для кластера Beowulf состоит из дополнений к ядру Linux, позволяющих создать единый образ системы, как для пользователей, так и для приложений. Пользователи запускают процессы на управляющем узле, система управления затем переносит (мигрирует) процессы на вычислительные узлы кластера.

Кроме дополнений к ядру система включает в себя набор библиотек для распределенных программ и различные утилиты для администрирования кластера.

Beowulf не поддерживает механизма контрольных точек и не имеет необходимой защиты от сбоев. В случае сбоя одного из узлов программа запускается заново. В будущих версиях разработчики планируют создать необходимую поддержку контрольных точек, также планируется возможность автоматического переключения на запасной мастер узел, что должно увеличить надежность работы всей системы [33].

Система специально создана для того, чтобы максимально использовать вычислительные возможности кластера. Улучшенные сетевые драйверы, специально оптимизированные библиотеки увеличивают производительность. Система может быть неэффективной для выполнения большого количества разнородных и небольших задач, так как имеет ограниченные возможности балансировки нагрузки. Тем не менее, систему можно отнести к наиболее эффективным для использования небольшим коллективом разработчиков.

4.2. Система Condor

Система Condor, в отличие от упомянутых выше систем, предназначена не для управления кластером, а для того, чтобы использовать простаивающие вычислительные ресурсы в сети рабочих станций. Основной принцип такой системы в том, что пользователь рабочей станции должен иметь возможность в любой момент времени использовать станцию в личных целях без каких-либо неудобств [9]. Система Condor имеет выделенный центральный узел, на котором работает управляющая часть системы. Специальный процесс постоянно контролирует активность на каждом из узлов, входящих в состав сети, и, в случае обнаружения активности пользователя, перемещает задачу на другую свободную машину. Condor периодически сохраняет задачу в

[2] CESDIS – The Center of Excellence in Space Data and Information Sciences

контрольных точках следующим образом. Программа пользователя собирается со специальной библиотекой. Система инициализирует процесс сохранения посредством посылки сигнала. Библиотека имеет обработчик такого сигнала, который производит сохранение всей информации в специальный файл, называющийся файлом контрольной точки. Этот файл содержит все необходимые данные для восстановления задачи и выполняемый код, осуществляющий такое восстановление. Таким образом, файл контрольной точки является обычной программой, которая при запуске восстанавливает исходный процесс.

Система интересна как первая в своем классе. Такой тип систем может быть полезен для больших компаний, особенно в выходные дни, когда большинство рабочих станций свободно для использования [20]. Однако такой принцип использования вычислительных ресурсов подходит только для очень ограниченного класса задач:

1. Возможно сохранение только одиночных процессов. Все процессы, порожденные посредством вызовов FORK и EXEC, не сохраняются.
2. Сохраняемый процесс не может использовать сигналы.
3. Никакие механизмы межпроцессного обмена, включая работу с сокетами, не поддерживаются.
4. Все операции ввода-вывода должны быть односторонними, т.е. либо только чтение, либо запись.

Следует отметить, что такая система должна обеспечивать максимальный контроль над запускаемыми программами с целью обеспечения защиты данных. Condor имеет очень серьезные недостатки с этой точки зрения. Так, практически отсутствует контроль над используемыми ресурсами, порождаемыми процессами и т.д.

4.3. Система EnFuzion

Система EnFuzion – это набор приложений для выполнения пакетных задач на кластерах с большим количеством узлов. EnFuzion была разработана а рамках исследовательского проекта Nimrod [1], выполнявшегося профессором Д. Абрамсоном в Австралийском университете Monash, в конце 90х годов. Затем система превратилась в коммерческий продукт и широко распространилась. Система способна распределять пакетные задания для кластеров, содержащих от нескольких десятков до нескольких сотен узлов [11]. Она имеет центральный выделенный узел, управляющий всеми остальными машинами в кластере. Система может функционировать как в пределах локальной сети, так и в масштабах Internet.

Система достаточно масштабируема, однако не имеет возможностей балансировки. С этим была связана очень интересная проблема, которая была замечена в Австралийском университете Monash во время использования данной системы [18, 19]. Система позволяет ограничивать количество

одновременно выполняющихся задач на узле. Достаточно объемные задачи просто занимали весь кластер и большому количеству пользователей, имеющих маленькие задачи, приходилось ждать. Авторы решили проблему путем написания CRON-скрипта, который периодически приостанавливал долго выполняющиеся задачи и позволял коротким быстро выполниться. Другой проблемой является пропускная способность сети, накладывающая ограничения на производительность системы в целом. Система трудна для администрирования. Все проблемы с несовместимостью программного обеспечения на узлах кластера должен решать администратор.

4.4. Система MOSIX

MOSIX – это набор дополнений к ядру Linux, позволяющий распределять процессы между узлами в составе кластера. MOSIX можно логически разделить на две части. Это миграция процессов и набор алгоритмов для эффективного динамического распределения ресурсов. Обе части реализованы на уровне ядра операционной системы и прозрачны для приложений. Миграция реализована следующим образом. Процесс разделяется на два контекста – контекст пользователя и системный контекст. Контекст пользователя содержит сегмент кода, стек, сегмент данных, содержимое регистров. Системный контекст содержит информацию об используемых ресурсах и стек ядра. Интерфейс между пользовательской и системной частями устанавливается на сетевом уровне. Пользовательская часть может мигрировать, системная часть жестко привязана к узлу, на котором был запущен процесс. Согласно [6, 7] время миграции состоит из двух компонент:

- фиксированная часть, когда происходит формирование нового образа процесса на удаленном узле;
- изменяющаяся часть, пропорциональная количеству передаваемых в связи с миграцией страниц памяти.

Взаимодействие компонент схематично изображено на рис. 2 [6, 7].

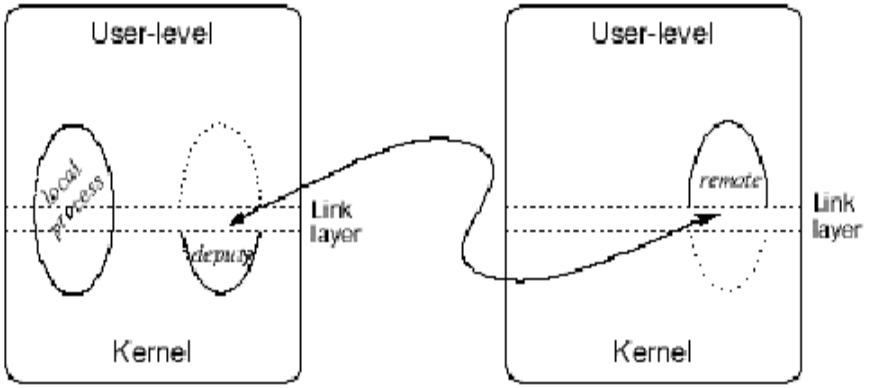

***Рис. 2** Миграция процесса в системе MOSIX*

MOSIX ориентирована на эффективное распределение ресурсов и обеспечение масштабируемости. Однако очевидным недостатком является большое количество накладных расходов, например, при обращении к сетевым ресурсам, осуществлении системных вызовов. Было даже предложено несколько интересных идей для уменьшения таких расходов. Например, мигрирующие сокеты и распределенные файловые системы [2], основную идею которых можно описать следующим образом. Если имеется процесс и некоторый ресурс, необходимый для данного процесса, то процесс может мигрировать на тот узел в кластере, на котором находится ресурс, вместо того, чтобы обращаться к нему удаленно. Однако такие способы могут лишь частично увеличить эффективность. Можно сказать, что MOSIX позволяет эффективно распределять ресурсы, но не позволяет их использовать максимально эффективно.

4.5. Система LoadLeveler

Данная система является модифицированной версией системы Condor. В отличие от своего предшественника, система имеет много дополнительных возможностей.

Система имеет встроенную подсистему статической балансировки, позволяющую разнообразными способами распределять поступающие от пользователя задачи. Система также имеет программный интерфейс пользователя (API), с помощью которого возможна реализация различных приложений, использующих возможности системы [14, 15]. Так, система

может использовать независимые внешние планировщики, широко используется планировщик MAUI, который будет описан позднее.

По использованию ресурсов все задачи делятся на классы. Система имеет предопределенный набор классов, который может дополняться или изменяться администратором. Пользователь, в таком случае, при запуске задачи указывает только ее класс.

Система имеет встроенную поддержку для пакетов MPI и PVM, для этого программы должны быть заново собраны со специальной библиотекой. Стоит отметить возможность системы осуществлять контрольное сохранение для параллельных задач.

Преимущества системы по сравнению с Condor также заключаются в больших возможностях с точки зрения безопасности. Появилась концепция доверительных хостов, контроль за используемыми ресурсами. Очень хороший графический интерфейс, позволяющий контролировать все аспекты работы системы [14]. Планировщик позволяет эффективно распределять нагрузку в кластере. Имеет смысл использовать LoadLeveler не только для простаивающих рабочих станций, но и как систему управления для полноценного кластера. В сочетании с миграцией процессов и контрольными точками даже для параллельных задач систему можно назвать почти универсальной.

4.6. Система LSF

Система LSF представляет собой набор приложений для администрирования и управления кластером рабочих станций. Система включает с себя подсистемы LSF Batch [24, 25, 26], LSF JobScheduler [29, 30], LSF MultiCluster, LSF Parallel [27], LSF Make, LSF Analyzer и основную подсистему LSF Base.

Рассмотрим архитектуру системы. Базовая система состоит из менеджера загрузки, называемого LIM, и сервера удаленного выполнения RES (сокращение от Remote Execution Server). LSF Base взаимодействует с операционной системой на каждом из узлов кластера и предоставляет единый интерфейс (API) для всех остальных компонент системы управления и пользователя, называемый Load Sharing LIBrary (сокращенно LSLIB). Менеджер загрузки собирает всю необходимую информацию с узла и обменивается ею с менеджерами, работающими на других узлах. Один из таких менеджеров является центральным и имеет в распоряжении информацию, собранную со всех остальных узлов, что схематично изображено на рис. 3 [24 – 30].

Система поддерживает самые разнообразные ресурсы, которые классифицируются следующим образом (указанные классы ресурсов могут пересекаться):

1. Ресурсы, доступные на всех узлах. Число процессоров на узле, количество оперативной памяти, статус узла, доступное дисковое пространство и т.д.

2. Специальные ресурсы, доступные только на определенных узлах. Например, наличие принтера, сервер сетевой файловой системы, определенная версия библиотеки и т.д.
3. Динамические ресурсы, изменяющие свое значение в процессе выполнения задачи.
4. Статические ресурсы.
5. Настраиваемые ресурсы, определенные пользователем или администратором.
6. Встроенные ресурсы.
7. Распределенные ресурсы. Данный тип ресурсов применяется по отношению ко всему кластеру. Любое изменение ресурса на каком-либо из узлов, так или иначе, может повлиять на другие узлы в составе системы или некоторого подмножества узлов. В качестве примера можно назвать ресурсы «пропускная способность сети», «групповая лицензия для программного обеспечения» или «сетевая файловая система».

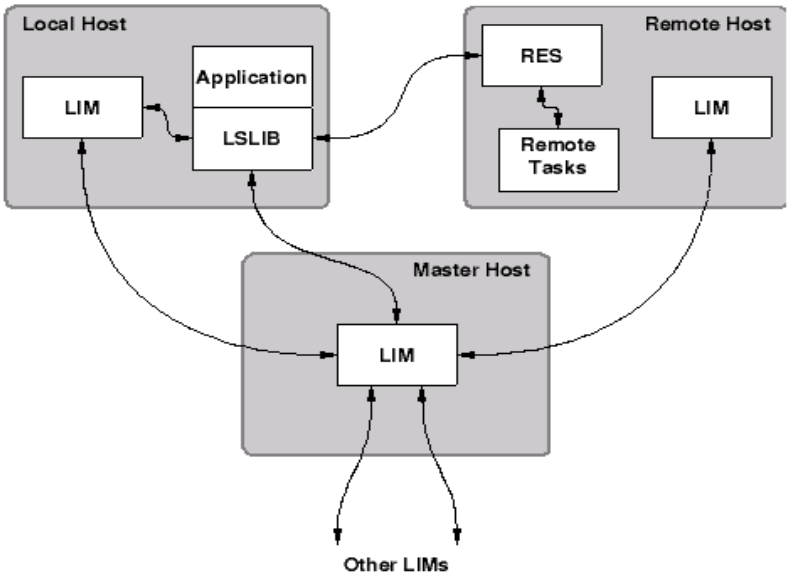

Рис. 3 Взаимодействие компонент в системе LSF

LSF JobScheduler в полной мере поддерживает все вышеперечисленные классы ресурсов при статическом и динамическом распределении нагрузки.

Система способна осуществлять три вида сохранения процесса в контрольных точках.

1. Сохранение на уровне ядра. В таком случае процесс сохранения прозрачен для приложений, все необходимые действия могут быть сделаны без каких либо изменений исполняемого кода программы.
2. Сохранение на уровне пользователя. Такой способ очень похож на уже описанный для систем CONDOR и LoadLeveler.
3. Сохранение на уровне приложения. В данном случае само приложение ответственно за свое сохранение и восстановление. Система только предоставляет необходимые функции из библиотеки.

В случае сбоя все остановившиеся задачи перезапускаются с момента последней контрольной точки. Стоит отметить, что все части системы, являющиеся центральными, например главный менеджер загрузки, могут быть в момент сбоя заменены новыми. В случае с LIM любой из остальных, подчиненных менеджеров, занимает место главного после его отказа.

Систему можно назвать универсальной. Мы указали только основную часть возможностей системы. Хочется отметить, что полный список гораздо шире. Например, система способна на работу в среде разнородных узлов, имеющих как различную архитектуру, так и под управление различных операционных систем. Впечатляет набор возможностей по указанию всевозможных ресурсов, в том числе и для параллельных программ. Подсистема LSF MultiCluster обеспечивает взаимодействие между несколькими кластерами в масштабах, например Internet. В этом случае используются различные методы защиты передаваемой информации, такие как шифровка, аутентификация и т.д. С другой стороны, все это делает систему гигантской как для установки, так и для поддержки. Одна документация для системного администратора занимает многие сотни страниц. Тем не менее, система может решить практически все требования конечного пользователя и использоваться в масштабах очень больших компаний.

4.7. Система Cleo

Система управления очередями Cleo предназначена для управления прохождением задач на многопроцессорных вычислительных установках (в том числе кластерных) [23]. Система является отечественной разработкой, созданной в НИВЦ МГУ. Система позволяет автоматически распределять вычислительные ресурсы между программами, управлять порядком их запуска, временем работы, получать информацию о состоянии очередей. Система достаточно проста как с точки зрения реализации, так и с точки зрения использования. Система написана на языке Perl и допускает подключение независимых модулей, расширяющих функции системы [23].

4.8. Системы PBS и PBS Pro

Системы PBS и PBS Pro [36] являются наследниками системы NQS [36], одной из первых систем управления кластерами. Иерархия этих систем управления схематично изображена на рис. 4.

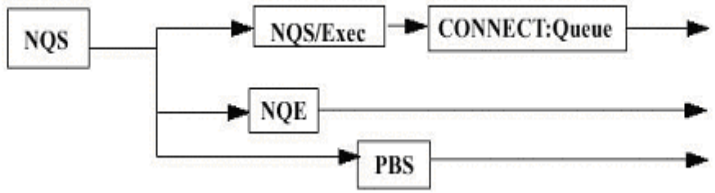

Рис. 4 Иерархия систем управления

Системы PBS и PBS Pro были созданы в подразделении американского космического агентства (NASA). Изначальным требованием к системе PBS помимо функциональности было удобство переноса системы на разные платформы. Система удовлетворяет стандарту POSIX 1003.2d и работает с такими операционными системами как Cray Unicos, SGI – IRIX, Sun – Solaris, Thinking Machines – CMOST, Intel – OSF/1-AD и другими.

Система имеет возможность настройки политики распределения заданий, и может работать с большим количеством (до нескольких тысяч) узлов.

Система может обслуживать несколько потоков очередей, разделенных как по архитектуре вычислительных узлов, так и по требуемым программе ресурсам. Определения ресурсов могут быть созданы администратором системы, и затем использоваться планировщиком при распределении заданий.

Система обеспечивает взаимодействие между несколькими кластерами через глобальную сеть, имеет графический интерфейс пользователя и WEB интерфейс для отправки заданий. Доступны разнообразные способы указания зависимостей между запускаемыми программами.

Системы PBS и PBS Pro могут быть удобны для организаций, имеющих разнородные кластеры с большим количеством узлов, в том числе суперкомпьютеров, и большое количество пользователей.

4.9. Планировщик MAUI

Планировщик является одной из основных частей системы управления. Он осуществляет всю работу, связанную с распределением задач на конкретные узлы в составе кластера. Такое распределение основывается на информации о статусе задачи, используемых ею ресурсах, и информации о доступных

ресурсах в системе. Многие системы управления имеют собственные, встроенные, возможности для такого распределения. С другой стороны сами алгоритмы распределения могут быть реализованы за пределами конкретной системы управления, используя только необходимую информацию, которую, в таком случае, предоставляет сама система посредством набора библиотечных функций (API).

В данном обзоре упомянем об одном из таких внешних планировщиков под названием MAUI. Планировщик MAUI был разработан в Maui High Performance Computing Center (MHPCC) в Мексике. Как говорилось выше, задача распределения ресурсов в общем случае принадлежит классу NP-полных задач [13, 32]. При большом количестве различных, в общем случае, несогласованных друг с другом требований достаточно трудно сделать правильных выбор тех узлов, на которых будет запущена программа. Одно из решений состоит в обеспечении возможностей настраивать поведение планировщика согласно некоторой политике распределения. MAUI был создан как раз для того, чтобы удовлетворить такие требования для администраторов и пользователей [21]. Рассмотрим некоторые из них.

1. Предположим, программы распределяются согласно приоритетам. В таком случае, рано или поздно, возникнет ситуация, при которой выбранная программа не сможет быть запущена по причине недостатка требуемых ресурсов (ярким примером может служить система EnFuzion). При этом программы, имеющие более низкий приоритет и требующие незначительных ресурсов, должны ждать. Планировщик MAUI способен разрешать такие ситуации и выбирать из списка ожидающих программ те, которые могут успеть выполниться. Выбор тоже не тривиален и является, в свою очередь, нетривиальной задачей распределения, правда уже второго уровня. Рекурсия в данном случае не используется, для распределения применяются более простые способы, например, выбирается первый подходящий узел, имеющий наибольший приоритет, или тот, который требует максимальное количество ресурсов. Такая политика распределения задач называется Backfill и поддерживается также многими встроенными планировщиками.
2. Другой крайней точкой является наличие большого количества доступных узлов, при этом возникает проблема выбора не программы, а узла. Существует несколько простых способов. Выбираются узлы, имеющие минимальные подходящие ресурсы, в таком случае наиболее производительные, в силу заданных требований, узлы остаются свободными. Выбираются те узлы, которые имеют наименее загруженный процессор. И, наконец, те, которые простаивают наибольший период времени.
3. Другие способы позволяют принимать решения на основе информации о пользователе или группе пользователей. Например, обеспечивая наилучшее качество сервиса, программы, принадлежащие заданной группе

пользователей, будут иметь высокий приоритет. Или наоборот, ограничивая количество задач для данного пользователя или группы пользователей. Также предусмотрена интересная возможность резервирования узлов. При этом указанные администратором узлы не будут использоваться под распределение задач до некоторого периода времени, который может определяться как явно, так и являться зависимым от состояния системы.

Возможности планировщика широки. Исходный код MAUI распространяется свободно, существует возможность внесения изменений и дополнений [21]. MAUI предоставляет возможность мониторинга системы: доступна различная информация о загрузке узлов, состояния задач, разнообразные статистики. Планировщик имеет независимый WEB интерфейс, созданы приложения, предоставляющие возможности администрирования и имеющие удобный графический интерфейс пользователя. Следует отметить, что в настоящее время MAUI может работать с системами LoadLeveler, PBS, PBSPro, Beowulf и другими.

5. Выводы

Мы рассмотрели несколько различных систем управления, каждая из которых имеет свои особенности, как с точки зрения функциональности, так и с точки зрения архитектуры. Из всех рассмотренных систем наибольший интерес для нас представляют две системы –Beowulf и Condor.

Система Beowulf (см. раздел 4.1) обычно устанавливается в университетах, научно-исследовательских институтах и небольших компаниях, нуждающихся в эффективном использовании кластера с относительно небольшим числом узлов для масштабируемых параллельных или распределенных вычислений.

Beowulf-кластер на базе коммуникационной сети SCI установлен в НИВЦ МГУ[3]. Для управления кластером используется созданная в НИВЦ система управления заданиями Cleo, описанная в разделе 4.7. Кластер Beowulf установлен и в ИСП РАН[4]. Кластер насчитывает 16 узлов, соединенных оборудованием Myrinet и имеет выделенный управляющий узел. В дальнейшем предполагается увеличить число узлов кластера. Кластер планируется использовать для выполнения параллельных (SPMD), распределенных и последовательных программ. Для обеспечения возможности эффективной эксплуатации кластера в ИСП РАН разрабатывается система управления кластером (см. ниже).

Система Condor обеспечивает наиболее простой способ использовать простаивающие рабочие станции. Такая система может принести пользу

[3] Научно Исследовательский Вычислительный Центр

[4] Институт Системного Программирования РАН

многим организациям, например, при компиляции и сборке больших программных проектов или проведении редких, но трудоемких вычислений. Использование рабочих станций в качестве свободного ресурса должно контролироваться пользователем. Например, система может использовать узел только после того, как пользователь закончил сеанс работы (logout) и освобождать узел при начале пользователем нового сеанса (login), возможно, с некоторой задержкой, необходимой для корректного завершения выполняемой программы. При этом по желанию пользователя система обязательно должна предоставлять информацию об использовании машины в его отсутствие.

Система подобная Condor нуждается в продуманном и надежном механизме защиты. Необходим максимальный контроль над запускаемыми задачами и использованием ресурсов. В случае сбоя или атаки под угрозой становится не только общая информация, относящаяся к выполняемым задачам, но и частная информация отдельных пользователей.

Конечно, отмеченные системы уступают по набору возможностей системе LSF, которая обладает практически всеми необходимыми возможностями (см. раздел 4.6). С другой стороны, система LSF имеет слишком много функциональности, что сильно отражается на ее компактности и удобстве использования. Большая часть возможностей системы, как правило, остается невостребованной. Лишь немногие компании могут позволить себе использовать такую дорогую систему, как LSF.

Сложной проблемой при выполнении параллельных программ является контроль над порождаемыми процессами. В большинстве случаев системы просто предоставляют необходимое количество узлов, действуя согласно пункту 2 указанного списка. Такой подход прост, но не эффективен. Программа не может динамически получать или освобождать узлы. В данной ситуации система MOSIX имеет явное преимущество, не делая различий между распределяемыми процессами.

Также важной задачей является восстановление параллельной программы после сбоя одного из узлов. Для распределенных систем запоминание согласованного глобального состояния является серьезной теоретической и главное практической проблемой.

Следует отметить необходимость реализации эффективных алгоритмов для передачи файлов. Многие системы используют либо сетевую файловую систему, либо стандартные приложения RSH, SSH и FTP. Рассмотрим пример. Пусть в некоторый момент времени необходимо отправить один файл на N узлов. Используя FTP, файл будет передан по сети N раз. Система может использовать альтернативные протоколы передачи файлов, способных передавать один файл многим получателям [34].

Согласно проекту система управления кластером, разрабатываемая в ИСП РАН, будет обеспечивать следующие возможности. Выполнение параллельных (распределенных и SPMD), последовательных и интерактивных задач.

Возможность указать в файле задания зависимость задач по результатам, управление через WEB интерфейс. Свойства системы можно выразить следующей таблицей (таб. 2).

Миграция процессов	Да
Пакетные задания	Да
Файл задания	Да
Зависимость задач по результатам	Да
Статус задачи	Да
Балансировка	Да
Задание ресурсов	Да
Задание приоритетов	Да
Интерактивные задачи	Да
Ограничение ресурсов	Да
Контрольные точки	Нет
Перекомпиляция	Нет
Защита от сбоев	Нет
Параллельные программы	Да
Монопольный режим	Да
Интерфейс	Да

Таблица 2. *Свойства системы разрабатываемой в ИСП РАН*

В дальнейшем планируется поддержка защиты от сбоев для всех классов задач и миграция процессов.

Литература

1. D. Abramson, R. Sosic, J. Giddy and B. Hall, "Nimrod: A Tool for Performing Parameterized Simulations using Distributed Workstations", The 4th IEEE Symposium on High Performance Distributed Computing, Virginia, August 1995.
2. L. Amar, A. Barak, A. Eizenberg, A. Shiloh, “The MOSIX Scalable Cluster File Systems for LINIX”.
3. Yair Amir, Baruch Awerbuch, A. Barak, R. Sean Borgstrom and Arie Keren, “An Opportunity Cost Approach for Job Assignment and Reassignment in a Scalable Computing Cluster”.
4. A. Avetisyan, S. Gaissaryan, O. Samovarov. “Extension of Java Environment by Facilities Supporting Development of SPMD Java-programs”. V. Malyshkin (Ed.): PaCT 2001, LNCS 2127, Springer-Verlag Berlin Heilderberg 2001, p. 175 – 180.
5. Mark A. Baker, Geoffrey C. Fox and Hon W. Yau, “Cluster Management Software”, Northeast Parallel Architectures Center 111 College Place Syracuse University New York 13244-4100 USA 16 November 1995.
6. A. Barak, O. La'adan, A. Shiloh, “Scalable Cluster Computing with MOSIX for LINUX”, the Hebrew University of Jerusalem.

7. A. Barak, O. La'adan, “The MOSIX Multicomputer Operating System for High performance Cluster Computing”.
8. The Beowulf project, www.beowulf.org.
9. A. Bricker, M. Litzkow, M. Livny, “Condor Technical Summary”, University of Wisconsin, 1991.
10. R. Buyya, “High performance cluster computing”, Volume 1, “Architectures and systems”, 1999.
11. EnFuzion 7.0 Administrator's Manual, Turbolinux.
12. Expect home page, http://expect.nist.gov/.
13. M.R. Garey, D.S. Johnson, “Computers and intractability: A guide to the theory of NP-Completeness”, 1979.
14. "IBM LoadLeveler User's Guide" IBM Corporation, 2001.
15. "IBM LoadLeveler Administration and Planning Guide" IBM Corporation, 2001.
16. S. Kannan, M. Roberts, P. Mayes, D. Brelsford, J.F Skovira, “Workload Management with LoadLeveler “, November 2001.
17. J.A. Kaplan, M.L. Nelson, “A comparison of Queuing, Cluster and Distributed Computing Systems”, NASA Langley Research Center June 1994.
18. C. Kopp, “Managing Cluster Computers. Parallelizing is the problem, flexibility is the key”, Dr. Dobb's Journal July 2000.
19. C. Kopp, “Supercomputing with TurboLinux enFuzion”.
20. M. Litzkow, M. Livny, “Experience With The Condor Distributed Batch System””, University of Wisconsin, 1991.
21. The Maui Scheduler, http://supercluster.org/maui.
22. R.P. Martin, A.M. Vahdat, D.E. Culler, and T.E. Anderson. Effects of Communication Latency, Overhead, and Bandwidth in a Cluster Architecture. //The 24th Annual International Symposium on Computer Architecture, 1997. http://now.cs.berkeley.edu/
23. http://www.parallel.ru/cluster, система управления заданиями Cleo.
24. Platform Computing Corporation, “LSF Batch User's Guide”, 6th edition August 1998.
25. Platform Computing Corporation, “LSF Batch Programmers's Guide”, 4th edition August 1998.
26. Platform Computing Corporation, “LSF Batch Administrator's Guide”, 6th edition August 1998.
27. Platform Computing Corporation, “LSF Parallel User's Guide”, June 2001.
28. Platform Computing Corporation, “Using LSF with Condor Checkpointing”, 2001.
29. Platform Computing Corporation, “LSF JobScheduler User's Guide”, Fourth Edition, January 2000.
30. Platform Computing Corporation, “LSF JobScheduler Administrator's Guide”, Third Edition, January 2000.
31. Psched, API Standards for Parallel Job/Resource Scheduling, 1998.
32. H. El-Rewini, T.G. Lewis, H.H. Ali, “Task scheduling in parallel and distributed systems”, 1994.
33. Scyld Computing Corporation, Technology Brief, April 2001 document #3720-2-1.
34. M. Sola, M. Ohta, Y. Muraoka, T. Maeno, “Scalable and Reliable Multicast File Transfer Architecture”.
35. NASA Ames Research Center, “Portable Batch System”, Requirements Specification.
36. http://pbs.mrj.com/, Portable Batch System Pro.