О р д е н а Л е н и н а
ИНСТИТУТ ПРИКЛАДНОЙ МАТЕМАТИКИ
имени М.В.Келдыша
Р о с с и й с к о й а к а д е м и и н а у к

В.Н. Коваленко, А.В. Орлов

Управление заданиями в распределенной среде и протокол резервирования ресурсов

Москва
2002 г.

УДК 519.68

В.Н. Коваленко, А.В. Орлов. **Управление заданиями в распределенной среде и протокол резервирования ресурсов.**

Рассмотрены вопросы управления заданиями в среде распределенных вычислений. Проанализированы возможные подходы к метадиспетчеризации. Один из них, реализуемый с применением резервирования, рассмотрен подробно. Приведен обзор средств резервирования с анализом возможности их использования при организации среды распределенных вычислений.

Ключевые слова: вычислительная Сеть, резервирование, управление заданиями

V.N. Kovalenko, A.V. Orlov. **Metascheduling in GRID and resource reservation protocol.**

Various aspects of the task scheduling in GRID are considered. The possible approaches to metascheduling are analyzed. One of them based on reservation is considered in detail. The review of local and global reservation tools with the analysis of their applicability in GRID design is given.

Key words: GRID, advanced reservation, metascheduling

Содержание

1. Введение

После длительного крена в сторону персональных компьютеров даже для решения задач, имеющих вычислительный характер, в последние годы наблюдается ренессанс коллективной обработки заданий, правда на качественно новом уровне. В течение прошедшего пятилетия сформировался и вышел на первый план новый подход к созданию мощных вычислительных систем в виде Среды Распределенных Вычислений (СРВ, в англоязычной литературе принят термин GRID). Идея состоит в том, чтобы интегрировать территориально разнесенные на любые расстояния (в масштабе региона, страны, Мира) компьютерные установки разных типов и назначений, используя в качестве средств коммутации высокопроизводительную сетевую аппаратуру и каналы связи. Важнейшая роль в концепции СРВ отводится системному программному обеспечению (ПО) – оно должно обеспечить качество интеграции, превращающее отдельные установки в единый мультипроцессор с функциональностью и моделью программирования обычных вычислительных комплексов. Фактически речь идет о создании полномасштабной операционной системы для открытой телекоммуникационно-вычислительной инфраструктуры, что определяет разнообразие вопросов обсуждаемых в рамках проблематики СРВ: управление заданиями, ресурсами, организация доступа к файлам, безопасность и т.д.

Развитие модели СРВ стимулируется появлением нового класса приложений, примерами которых служат задачи распределенного управления, обработки и хранения сверхбольших объемов информации. Используя унифицированные протоколы СРВ, можно организовывать виртуальные комплексы обработки экспериментальных данных разной природы, которые в реальном времени будут собирать данные с датчиков, производить первичный анализ, вырабатывать решения и выдавать управляющие воздействия. Такого рода комплексные задачи уже возникли в биологии (изучение генома человека), ядерной физике (анализ высокоэнергетических ядерных реакций), экологии (мониторинг и моделирование окружающей среды).

Работы по СРВ были начаты в нескольких университетских лабораториях США [1], [2], сейчас же это направление оценивается как одно из самых перспективных в сфере информационных технологий, а соответствующие исследования и разработки ведутся во всем мире. К настоящему времени накоплен существенный задел в виде архитектурных решений, протоколов, реализован базовый слой программного обеспечения СРВ, проведены масштабные демонстрации и эксперименты на вычислительных стендах по глобальным телекоммуникациям [3].

Стандартом де-факто стала система Globus [4], в которой реализованы протоколы сетевых коммуникаций, дистанционного запуска и управления заданиями, мониторинга, а также протоколы безопасности и информационного

обслуживания. На базе системы Globus развернуты несколько корпоративных СРВ [5-7]. Параллельно идет создание приложений и использование уже имеющихся возможностей для решения прикладных задач [8].

Работы, ведущиеся в ИПМ РАН по тематике СРВ, сконцентрированы на одной из важных задач общего направления - создании средств управления заданиями. В Институте организован стенд СРВ, включающий две территориально разнесенные площадки, на каждой из которых работают сетевые кластеры из нескольких рабочих станций разной архитектуры. На этой базе реализован первый программный проект - метадиспетчеризации заданий [9], [10]. Заложенная в нем схема управления, хотя и обеспечивает требуемую функциональность, использует простейшие алгоритмы и, как следствие, не слишком эффективна и плохо масштабируема. Достигнута, однако, другая существенная цель – весь проект выполнен в рамках протоколов системы Globus и является ее расширением. В данной работе мы рассматриваем дальнейшее развитие, направленное на оптимизацию, и, в частности, один из перспективных подходов – предварительное резервирование (advance reservation).

2. Архитектура управления заданиями СРВ

Основное назначение СРВ – дать возможность большому (теоретически неограниченному) коллективу пользователей, которые независимо работают в разных точках глобальной сети, запускать задания на ресурсы вычислительных комплексов (ВК), образующих СРВ в режиме свободного подключения/отключения. Если же прикладная задача распараллелена, то для нее из СРВ может быть получено любое количество ресурсов.

Уточним понятие задания. Под заданием для СРВ понимается исполняемый файл типа .EXE или файл командной оболочки shell. С ним "в комплекте" находятся описание ресурсов (ресурсный запрос) – например, объем памяти, тип платформы, - которые необходимы для его счета, и предполагаемое время выполнения. Базовое ПО СРВ (здесь и далее имеется система Globus) содержит пользовательский интерфейс, позволяющий запустить задание на любой ВК СРВ. Для иллюстрации приведем команду запуска задания пользовательского интерфейса системы Globus.

```
globusrun -r supercomputer.keldysh.ru/jobmanager-pbs \
'&(executable=/home/user/script.sh)(maxWallTime=100)\
(maxMemory=10)'
```

Здесь в опции -r задан сетевой адрес ВК для запуска, далее указаны исполняемый скрипт и ресурсный запрос на время и память. Отметим, что при

использовании базовых средств для запуска, выбор места выполнения возлагается на пользователя.

СРВ является средой коллективного пользования. Известно, что для операционных сред такого типа, в особенности, если они достаточно масштабны, необходимым элементом является система управления. Мы будем рассматривать систему **диспетчеризации**, которая выполняет функции распределения заданий по ресурсам СРВ. Если допустить, что автоматическая диспетчеризация отсутствует (как в базовом ПО СРВ), то при любом объеме ресурсов возможен, с одной стороны, простой их части на некоторых ВК (несмотря на наличие готовых к запуску заданий), а с другой стороны возможна перегрузка отдельных ВК, на которые поступает неконтролируемый поток заданий от распределенных пользователей, не координирующих, естественно, свои действия. Последний вариант недопустим, так как всякий ВК имеет конечные объемы виртуальной памяти и дискового пространства. Если поток заданий не регулируется, то считающиеся задачи начнут аварийно завершаться по причине нехватки этих ресурсов.

Как отражено в названии, **СРВ является распределенной средой** и должна удовлетворять требованиям надежности и масштабируемости. Отсюда следует, что система управления СРВ также должна быть распределенной. Свойство распределенности уже заложено в архитектурном решении для СРВ, на которое рассчитано базовое ПО. Это проявляется в наличии системы управления на каждом из ВК, формирующих СРВ. В качестве ВК в СРВ естественно применять высокопроизводительные архитектуры – многопроцессорные установки, специализированные машины баз данных, архивы данных большой емкости с быстрым доступом, графические комплексы рендеринга. Такие архитектуры могут технически реализовываться разными способами, в том числе методами кластеризации с помощью коммуникационной аппаратуры в локальной сети. Таким образом, под ВК понимается, как правило, совокупность разнотипных ресурсов (процессорных, сетевых, оперативной и постоянной памяти) большого объема.

В качестве систем управления ВК применяются системы пакетной обработки заданий (СПО), регулирующие загрузку на ресурсы и распределяющие их между заданиями. Регулировка загрузки осуществляется путем буферизации поступающих заданий в буферную очередь, а запуск их на счет контролируется процедурой планирования. Следует отметить, что СПО имеют достаточно длинную историю применения для управления обработкой на мощных установках. Архитектура СРВ отражает сложившееся положение, используя СПО в качестве нижнего уровня управления.

Система Globus поддерживает программный интерфейс с несколькими типами СПО - Condor, PBS, Loadleveler, LSF и т.д. Весь процесс дистанционного запуска задания средствами системы Globus начинается с передачи задания по сети от пользовательского интерфейса на сервис GRAM вычислительной установки. Здесь проводится контроль доступа (процедура

взаимной аутентификации пользователя и GRAM). Далее задание вводится в СПО (командой пользовательского интерфейса СПО – Qsub) и помещается в очередь. В общем случае очередь СПО формируется из заданий внутренних пользователей, которые вводят их непосредственно через интерфейс СПО, и внешних, применяющих интерфейс СРВ.

Порядок обработки очереди, а следовательно, и время ожидания заданий, и то, на каких ресурсах они будут выполняться, полностью определяется СПО. Для современных СПО характерна высокая степень конфигурируемости процесса обработки. В среде СРВ это позволяет выдержать важный принцип – сохранить права владельца ВК на собственные ресурсы. Администратор СПО может настроить обработку таким образом, чтобы выделять их внешним пользователям в такое время и в таком объеме, в каком он считает нужным.

2.1. Диспетчеризация заданий в СРВ

Управляющая система диспетчеризации заданий для СРВ, по крайней мере, пока может основываться на идеях, которые были отработаны в СПО. Полем ресурсов СРВ заведует сервер – метадиспетчер. На него поступает поток заданий от всех пользователей. Метадиспетчер буферизует задания в очередь (спул), далее, руководствуясь той или иной политикой, находит им ресурсы и перенаправляет их на тот или иной ВК, то есть в СПО ВК. Иначе говоря, отличие метадиспетчера от СПО заключается в том, что задания распределяются не по вычислительным машинам (процессорам), а по ВК. Таким образом, в архитектуре СРВ можно выделить два уровня организации и, соответственно, два уровня управления заданиями: 1) локальный, соответствующий локальной сети, в котором обработкой заданий управляет СПО и 2) глобальный, который объединяет множество ВК локального уровня, и в котором заданиями управляет метадиспетчер (рис. 1).

В похожести функций СПО и метадиспетчера скрыто существенное структурное отличие: в СРВ мы имеем два управляющих уровня и общая эффективность управления в значительной степени определяется способами их взаимодействия. Дело осложняется тем, что СПО изначально проектировались как изолированные системы. Они располагают интерфейсом пользователя для запуска и управления заданиями, а также административным интерфейсом для конфигурирования, управления очередями и машинами. Этих средств конечно мало для поддержки какого-либо оптимального планирования на глобальном уровне, однако в работе [10] показано, что возможна метадиспетчеризация, позволяющая выбрать адекватный ресурсному запросу задания ВК и обнаружить простаивающие машины. Ясно,

однако, что необходимы более развитые интерфейсы и, что более важно, более мощные механизмы управления в самих СПО.

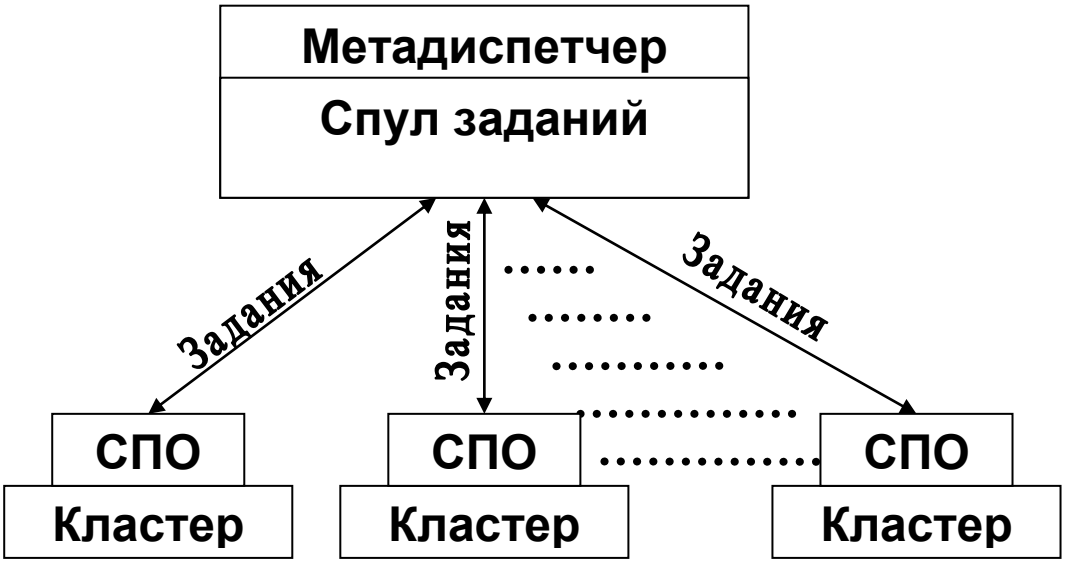

Рис. 1 Среда распределенных вычислений

2.2. Планирование процесса обработки в СРВ

Ключевым моментом диспетчеризации является этап планирования. На этом этапе для каждого задания из очереди в определенный момент должно быть принято решение, на какой из ВК следует его направлять. Рассмотрим далее, каким образом может осуществляться планирование на уровне СРВ. Должны учитываться, по крайней мере, два обстоятельства.

1. Задание следует запускать только на те ВК, которые располагают ресурсами, необходимыми для его выполнения. Пример: задание не следует запускать на ВК с предельным объемом памяти в 256 Мб, если ему требуется 1 Гб памяти. Отбор подходящих по объему ресурсов ВК можно выполнить имеющимися в системе Globus средствами, и такая реализация сделана в [10]. Команда запуска задания имеет опции для определения количества требуемых ресурсов (хотя список возможных типов ресурсов несколько ограничен). А в состав системы Globus входит информационный сервис (ИС), обслуживающий распределенную базу данных о характеристиках ВК СРВ. Отбор допустимых ВК производится поисковым запросом к ИС.

2. От выбора того или иного ВК существенно зависит время обработки задания, которое складывается из нескольких составляющих: времени ожидания в очереди СПО, собственно времени счета и времени доставки файлов задания. Что касается времени счета, то оно задается в ресурсном запросе (процессорный ресурс). Напротив, время ожидания в рамках современного интерфейса СПО остается непрогнозируемой извне величиной, которая зависит от конфигурации и состояния СПО. Действительно, для того, чтобы оценить возможное время ожидания, нужно учесть имеющиеся в очереди задания, их приоритеты, загрузку ресурсов, режим их распределения и т.д. Иначе как путем моделирования будущего это, по-видимому, сделать нельзя. Задание можно отдать на счет в разные СПО, но от того, насколько удачен выбор, будет зависеть суммарное время обработки. Поэтому вопрос эффективной диспетчеризации в большой степени определяется возможностью предсказания времени старта.

2.3. Возможные подходы к метадиспетчеризации

Простейшим из возможных подходов к организации работы метадиспетчера является жесткое выделение (резервирование) ресурсов ВК для глобального использования. То есть, администраторы на местах отдают часть своих ресурсов на некоторое время под полный контроль метадиспетчера. Эти ресурсы будут недоступны для локального использования на время внешнего резервирования, и будут простаивать, если метадиспетчер недозагрузит их. Имеющаяся статистика показывает [11], что при таком подходе средний уровень использования ресурсов не превышает 25%. Из-за чего же происходят потери? Главным образом из-за нескоординированной работы глобального и локального планировщиков. Например, несмотря на простаивающие ресурсы, локальные приложения не могут их использовать, так как в данный момент эти ресурсы забронированы для метазаданий. Ситуация может быть и обратной: метадиспетчер задерживает свою очередь, в то время как на локальном узле имеются свободные ресурсы.

Таким образом, приходим к выводу, что требуется совместная работа локального и глобального уровня планирования, учитывающая их политики и принципы предоставления ресурсов. Такое взаимодействие должно отражаться не в выделении фиксированного количества ресурсов в фиксированное время, а только под задания, уже стоящие в очереди метадиспетчера, в таком объеме, который им необходим, и в то время, когда они будут свободны. Но, пообещав ресурсы, СПО должна гарантировать, что в указанное ей время они будут доступны. Т.е. она должен зарезервировать ресурсы для конкретного задания. Такие гарантии предоставления

запрашиваемых ресурсов в будущем называются **предварительным резервированием** (advance reservation).

3. Планировщик Maui

Возможность практического применения идеи резервирования ресурсов упирается в поддержку этого механизма системами пакетной обработки. На сегодняшний день ни одна из них средств резервирования не имеет – кроме планировщика Maui [12].

Maui, разработанный в Maui High Performance Computing Center (MHPCC, www.mhpcc.edu), не является полной СПО, а только одной из компонентов – планировщиком. Зато он может подключаться к разным СПО, имеющим модульную архитектуру – модульность позволяет заменять компоненты в рамках заданных интерфейсов. К числу СПО, для которых интерфейсы с Maui уже реализованы, относятся широко распространенные PBS [13] и Loadleveler [14]. При использовании PBS, например, довольно просто поменять стандартный планировщик (FIFO) на Maui.

Как и для всех других планировщиков, применяемых в СПО, функции Maui следующие: при заданном множестве готовых к исполнению заданий (из очереди) и для текущего состояния ресурсов определить очередность запуска заданий на счет и найти подходящие ресурсы для тех из них, которые можно запустить в данный момент. В этих рамках Maui реализует новые механизмы, основанные на “предвидении будущего” и направленные на оптимизацию управления. Помимо того, в Maui есть ряд новшеств, отличающих его от предшественников. Это:

- Алгоритмы обратного заполнения (backfill) и справедливого распределения ресурсов (fairshare) для повышения эффективности системы и уменьшения времени ожидания в очереди.
- Система автоматического определения приоритетов заданий, позволяющая давать ключевым пользователям преимущество при распределении ресурсов.
- Улучшенная диагностика проблем с заданиями, узлами и программными компонентами СПО.
- Расширенный спектр статистики. С помощью Maui администратор может получать полную историческую и текущую информацию о заданиях, очередях, планировщике, состоянии системы и т.п.
- В Maui есть возможность моделирования работы СПО, с помощью которой можно проверить настройки планировщика "в деле". Это позволяет подобрать конфигурацию системы, наиболее полно отвечающую требованиям конкретной производственной обстановки.

У Maui есть еще одно существенное отличие: обычные планировщики не имеют собственных командных интерфейсов – они им не нужны. Maui же, полностью реализуя функции резервирования, располагает и соответствующим интерфейсом для внешнего или, так называемого, административного резервирования. Стоит отметить, что в такой изолированности резервирования от остальной части СПО, есть недостатки – вообще-то необходимо координировать действия по резервированию (осуществляемые через Maui) и управления заданиями, очередями и узлами, но это, наверное, дело будущего.

3.1. Аппарат резервирования Maui

Понятие резервирования формализуется в Maui соответствующим типом объектов. Объект "резервирование" состоит из:

- ACL (Access Control List, список контроля доступа) – набор параметров, руководствуясь которым планировщик определяет, для каких заданий доступны зарезервированные ресурсы.
- Список зарезервированных ресурсов.
- Время действия резервирования.

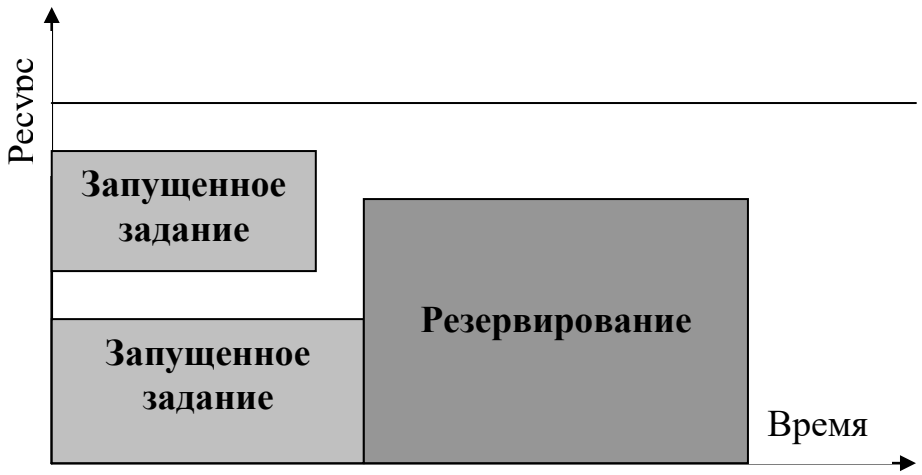

Рис. 2 Резервирование в Maui

На уровне реализации каждое резервирование физически представляет собой запись в базе данных Maui. Объект резервирование привязан, с одной стороны, ко времени, причем к будущему. С другой стороны, интерпретация резервирований предполагает рассмотрение состояний ресурсов на моменты начала резервирований. Изображается это в виде временной развертки (time line) состояний ресурсов: для каждого ресурса на оси времени ставятся метки,

определяющие, какому заданию в этот момент ресурсы могут быть доступны (рис. 2). Совокупность некоторого количества меток и образуют одно резервирование (т.к. зарезервировать можно целый комплекс ресурсов, и все они будут принадлежать одному резервированию).

Основной инструмент резервирования - команда setres. Ее выполнение разрешено только пользователям с правами администратора Maui. Это впрочем, не означает, что она не может выполняться автоматической системой управления, обладающей такими правами.

setres [ARGUMENTS] <RESOURCE_EXPRESSION>

ARGUMENTS: <имя резервирования>, <ACL-атрибуты>, <атрибуты времени жизни резервирования>, <резервируемые ресурсы>

<имя резервирования>: -n <NAME>

<ACL-атрибуты>: -a <ACCOUNTLIST>
-g <GROUPLIST>
-l <CLASSLIST>
-u <USERLIST>
-q <QOSLIST>
<атрибуты времени жизни резервирования>: -d <DURATION>
-e <ENDTIME>
-s <STARTTIME>

<резервируемые ресурсы>: -r <RESOURCE_DESCRIPTION>

В результате выполнения команды setres порождается объект типа “резервирование” с временем жизни, которое задается аргументами STARTTIME, ENDTIME и DURATION. Этот набор избыточен, поэтому достаточно указывать только один из двух последних параметров. Объект можно поименовать с помощью аргумента NAME. Если этот аргумент отсутствует, то имя присваивается автоматически.

Заметим, что резервирование (предварительное!) производится под задание, которого еще нет в СПО, поэтому должен быть механизм, позволяющий связать задание со сделанным резервированием. Объект резервирование обладает списком контроля доступа ACL, и именно по нему происходит привязка. Зарезервированные ресурсы могут получить только такие задания, которые имеют хотя бы один из параметров: GROUPLIST, USERLIST, ACCOUNTLIST, CLASSLIST и QOSLIST, - совпадающий по значению с соответствующими атрибутами ACL резервирования. То есть привязка может осуществляться по регистрационному имени пользователя, его группе, классу и качеству обслуживания, а также с помощью особой метки –

регистрационному имени (account) проекта. Возможности настройки списков доступа (ACL) в Maui довольно велики и, в частности, можно добиться того, чтобы зарезервированные ресурсы были доступны только для определенного задания, причем даже еще не поступившего в очередь СПО (именно это требуется для метадиспетчеризации).

Maui позволяет зарезервировать 4 наиболее важных типа ресурсов:

[PROCS=<INTEGER>:][MEM=<INTEGER>:][DISK=<INTEGER>:]
[SWAP=<INTEGER>:]

Определяется количество ресурсов, отводимом на одну задачу – в задание может входить несколько параллельных задач, каждая из которых выполняется на одном узле ВК. Не следует путать "задание" и "задача" (в документации Maui используются соответственно "job" и "task"). Каждое задание может состоять из нескольких задач. Задача может выполняться только на одном узле и является как бы квантом задания. Когда осуществляется резервирование под задание, то в запросе указываются ресурсы, необходимые для одной задачи и количество задач. Таким образом, все задачи представляются однотипными, по крайней мере, требующими одинаковое количество ресурсов. Далее мы ограничимся рассмотрением только однозадачных заданий.

В Maui предусмотрены средства для операций с уже существующими резервированиями. С помощью команды showres можно получить информацию о том где, для каких заданий, на какое время, и сколько ресурсов зарезервировано, причем доступен поиск как по резервированиям, так и по узлам. Существует также команда releaseres для отмены созданных резервирований.

3.2. Алгоритм планирования Maui

Как и многие другие планировщики, Maui работает итерационно, т.е. перемежая процесс планирования с ожиданием или выполнением внешних команд. Каждый цикл начинается при осуществлении одного из следующих событий:

- меняется состояние задания или ресурса
- достигнута граница резервирования
- получена внешняя команда
- с начала предыдущего цикла прошло время, определенное как максимальное.

В процессе планирования применяется еще один, внутренний тип резервирования Maui – job резервирование. С точки зрения реализации оно мало отличается от описанного выше административного резервирования. Используется оно в двух целях.

Во-первых, для обеспечения доступности ресурсов под выполняющиеся задания. Как только задание запускается, то сразу, на все время исполнения (walltime), создается job резервирование, закрывающее доступ к ресурсам, необходимым для этого задания, всем, кроме него самого. Может быть, во время работы все эти ресурсы и не будут использоваться, однако как только они потребуются заданию, оно гарантировано сможет их получить. Когда задание завершается (возможно, раньше, чем через walltime) job резервирование отменяется, и ресурсы становятся свободными.

Во-вторых, job резервирование применяется для того, чтобы задания с более высокими приоритетами не были задержаны запуском менее приоритетных, что может случиться при работе алгоритма backfill. В общем виде процесс запуска заданий выглядит так: до тех пор, пока это возможно, упорядоченные по приоритетам задания берутся из очереди и запускаются (попутно создается job резервирование); как только очередное задание нельзя запустить из-за нехватки ресурсов для него, определяется ближайшее время, в которое можно будет произвести запуск, и начиная с него создается job резервирование затребованных ресурсов на время walltime.

После того как сделано некоторое, конфигурируемое количество резервирований, начинает работу собственно backfill. Он выясняет, какие узлы и на какое время, начиная с текущего, свободны (это как раз определяется по сделанным job резервированиям). После этого свободные узлы объединяются в окна. Суммарная "ширина" окон может оказаться больше количества свободных в данный момент узлов, т.к. некоторые узлы могут входить в несколько окон (рис 3). Затем из всех окон выбирается одно, как правило, самое широкое (на рис. 3 – Окно 1). Среди всех оставшихся заданий выбирается наиболее точно удовлетворяющее этому "окну" задание и запускается. Если есть возможность, то запускается не одно задание. Так как при запуске заданий алгоритмом backfill учитываются сделанные job резервирования, то соответствующие им

Рис. 3 Backfill окна

задания не будут задержаны.

Основным отличием job резервирования от административного является то, что job резервирование осуществляется самим планировщиком и им же отменяется перед началом очередного шага планирования. Причем картина резервирований может отличаться от предыдущего шага планирования весьма существенно. Это происходит в случае, когда поступают новые задания, меняющие порядок в очереди или освобождаются ресурсы. В отличие от административных, job резервирования не гарантируют запуск задания в указанное время. Таким образом, job резервирования (не для запущенных заданий) являются всего лишь метками, позволяющими корректно учесть в алгоритме backfill приоритеты заданий. Административные и job резервирования существуют параллельно, т.е. job резервирования могут накладываться на административные, если конечно задание, под которое делается job резервирование, удовлетворяет ACL административного.

Рассмотрим теперь один цикл работы Maui. Он состоит из следующих шагов.

1. Получая сведения от СПО о машинах, стартовавших/закончившихся заданиях, конфигурации системы, Maui обновляет свою внутреннюю информацию о состоянии ресурсов.
2. Корректируются сделанные резервирования, исходя из обновленной информации о занятости узлов. Конкретнее – отменяются job резервирования, соответствующие уже завершившимся заданиям. На этой же фазе запускаются те ожидающие задания, время действия job резервирований для которых уже наступило.
3. Далее из всего количества заданий, стоящих в очереди СПО, отбираются те, которые готовы к запуску в настоящий момент. При этом принимаются во внимание такие аспекты, как состояние задания (задержано оно или нет), достаточно ли ресурсов всего кластера для запуска этого задания и т.п. После этого из получившегося списка отсеиваются задания в соответствии с политикой кластера.
4. Получившееся множество заданий упорядочивается в соответствии с приоритетами, которые назначаются исходя из ряда условий. В частности, принимаются во внимание запрашиваемые ресурсы, принадлежность задания тому или иному пользователю, историческая информация (кто сколько заданий запустил, чтобы избежать ситуации, когда кластер "оккупирован" кем-то, и недоступен для всех остальных).
5. Теперь в соответствии с полученным упорядоченным списком производится планирование. Задания рассматриваются, начиная с самого приоритетного. Последовательно получая задания из списка, Maui запускает их до тех пор, пока это возможно (конечно же, создавая job резервирования, обеспечивающие их дальнейшую работу). Стоит заметить, что тут ограничивающими факторами выступают job резервирования

запущенных заданий и административные резервирования, запрещающие доступ к ресурсам для заданий, не соответствующих их ACL. Если текущее задание с наибольшим приоритетом не может быть запущено из-за недостатка свободных ресурсов на время walltime, то для него определяется момент, когда это сделать будет возможно, и, начиная с него, создается job резервирование. Процесс перебора заданий и запуска/резервирования продолжается до тех пор, пока не будет сделано некоторое заданное конфигурацией Maui количество резервирований.

6. Наконец, оставшиеся задания (т.е. те, под которые не сделано резервирований) алгоритм backfill пытается пристроить на оставшиеся свободными ресурсы, но так, чтобы не "залезать" на сделанные резервирования.

Резервирование – достаточно общее понятие, допускающее различные варианты воплощения. Некоторые из решений в Maui выглядят не бесспорными, особенно в связи с метадиспетчеризацией. Так, например, задание, под которое сделано резервирование, может быть выполнено на других ресурсах, если они вдруг освободятся раньше зарезервированных. На первый взгляд такой подход представляется естественным. Действительно, зачем простаивать свободным ресурсам, а заданию находиться лишнее время в очереди? Существуют, однако, причины, по которым преждевременный запуск может быть невозможен. Например, как правило, перед началом счета на исполнительную машину должны быть доставлены входные файлы. Возможны три пути решения этой проблемы: 1) разрешить запуск заданий раньше времени; 2) запретить запуск заданий раньше времени; 3) добавить возможность установки на задание флага, разрешающего/запрещающего преждевременный запуск. В Maui выбран первый вариант, но вполне вероятно, что предпочтительным окажется третий, правда для него потребуется расширение интерфейса.

С другой стороны, некоторые аспекты резервирования представляются избыточными. Разрешается делать несколько резервирований на одни и те же узлы в одно и то же время. Доступ к ним будет разрешен заданию при условии, что его атрибуты удовлетворяют ACL каждого из этих резервирований. Это свойство позволяет гибко управлять доступом к ресурсам, однако таит в себе и потенциал для накладок. Коллизия возникнет, когда несогласованно создаются два резервирования на одни и те же узлы для двух разных заданий. В этом случае запуститься не сможет ни одно. При метадиспетчеризации производится внешнее (административное) резервирование под конкретные задания, но явного средства для этого нет. Поэтому требуется ввести механизм и дополнительные соглашения, предотвращающие возникновение конфликтных ситуаций при использовании резервирований.

3.3. Использование предварительного резервирования для метадиспетчеризации

Как было показано, предварительное резервирование полезно на локальном уровне управления: с его помощью обеспечивается доступность ресурсов для запущенных заданий, поддерживается алгоритм backfill и т.д. Возвращаясь к метадиспетчеризации, выскажем мнение, что ценность подхода, предложенного в Maui, не ограничивается собственно резервированием. Интересен сам механизм, которым резервирование поддерживается. Он основан на предсказании будущих состояний СПО, и, следовательно, позволяет оценить возможное время старта заданий, а это как раз тот параметр, который нужен для планирования на глобальном уровне управления. При наличии оценки времени старта метадиспетчеру не нужна детальная информация о СПО – загрузке отдельных процессоров, количестве заданий в очередях и т.п. Диспетчеризация может производиться тогда по следующей схеме:

1. Пользователь посылает задание и ресурсный запрос метадиспетчеру.
2. Метадиспетчер опрашивает обслуживаемые ВК, посылая им ресурсный запрос задания и получает в ответ возможные времена запуска.
3. Метадиспетчер выбирает ВК, например, по ближайшему времени запуска, и посылает туда запрос на резервирование ресурсов, причем резервирование должно быть создано таким образом, чтобы воспользоваться зарезервированными ресурсами могло только это задание.
4. Если резервирование сделано, задание посылается в очередь СПО. По-видимому, это должна быть отдельная очередь, и СПО должна гарантировать, что, когда наступит время резервирования, задание сможет им воспользоваться. В том случае, если ресурсы, необходимые для запуска задания освободятся раньше времени начала резервирования, Maui разрешает запустить задание.

Разумеется, это лишь принципиальная схема, которая не учитывает ряд моментов, но она является существенным продвижением по сравнению с тем, что можно сделать на интерфейсах традиционных СПО.

Для реализации данной схемы метадиспетчеризации требуются две интерфейсные функции, которые должна поддерживать СПО:

Функция оценки времени старта {
Входные параметры: ресурсный запрос, время счета;
Выходные параметры: список возможных времен старта}

Функция резервирования ресурсов{

Входные параметры: ресурсный запрос, время начала резервирования, длительность резервирования;
Выходные параметры: результат попытки создания резервирования}

Может показаться, что достаточно иметь только первую из этих функций. Это не так: необходимость резервирования ресурсов возникает из-за того, что интервал времени от момента получения оценки и выбора ВК до момента запуска задания на счет, когда СПО реально выделяет ресурсы заданию, может быть большим. Состояние СПО может измениться – появятся новые задания с более высоким приоритетом, – и ресурсы заданию не достанутся. При выполнении же резервирования старт задания гарантируется.

4. Интерфейс удаленного резервирования

На данный момент реальное положение дел с функциями для поддержки метадиспетчеризации следующее. Функция резервирования входит в состав командного интерфейса Maui, что же касается второй функции – оценки времени старта, то она реализуется как расширение базовой функциональности Maui в связи с проектами Silver [12] и Ursala [11], но ее спецификации до сих пор не опубликованы.

В любом случае, разработчикам метадиспетчера необходимо решить еще один круг вопросов. И командный, и программный (API) интерфейс Maui предназначен для работы в локальной среде, между тем как метадиспетчеризация выполняется в среде глобальной, свойства которой (открытость и масштабность) предполагают принципиально иные решения по протоколам связи, безопасности и информационной службе. В этом плане представляют интерес предложения [15] по глобальному протоколу резервирования, согласованные с протоколами системы Globus. Более того, имеется частичная реализация API удаленного резервирования в комплексе GARA [16].

4.1. GARA (Универсальная Архитектура Размещения и Резервирования).

GARA предоставляет средство удаленного резервирования, унифицированное по ресурсам разного типа: сетевых, процессорных, дисковых и т.д. В отличие от Maui, GARA распространяет возможность резервирования на сетевые ресурсы, позволяя зарезервировать, например, определенную пропускную полосу канала связи.

Аббревиатура GARA обозначает два понятия: собственно архитектуру, то есть принцип организации удаленного резервирования, и API для создания резервирований. Архитектура GARA, изображенная на рис. 4, содержит три уровня API. Верхние уровни осуществляют передачу запроса по сети, опираясь на протокол аутентификации системы Globus. Нижний уровень передает запрос компоненте, осуществляющей резервирование, по терминологии GARA - локальному менеджеру ресурсов.

Рис 4. Архитектура GARA

API GARA имеет две интересные особенности. Во-первых, возможность создания как предварительных резервирований, так и немедленных, т.е. начинающихся с текущего времени. Во-вторых, один и тот же API используется при резервировании всех типов ресурсов и, в основном, не зависит от типа, что упрощает процесс программирования.

API GARA можно рассматривать как механизм удаленного вызова процедур для взаимодействия с менеджером ресурсов, непосредственно контролирующим резервирование. Схема работы удаленного вызова представлена на рис. 5. Из этой схемы видно, что программа связывается с менеджером ресурсов не напрямую, а через Globus gatekeeper. Gatekeeper осуществляет три важные функции: аутентификацию пользователя;

авторизацию, проверку того, имеет ли право этот пользователь делать резервирования; наконец, запускает сервис, управляющий взаимодействием с менеджером ресурсов.

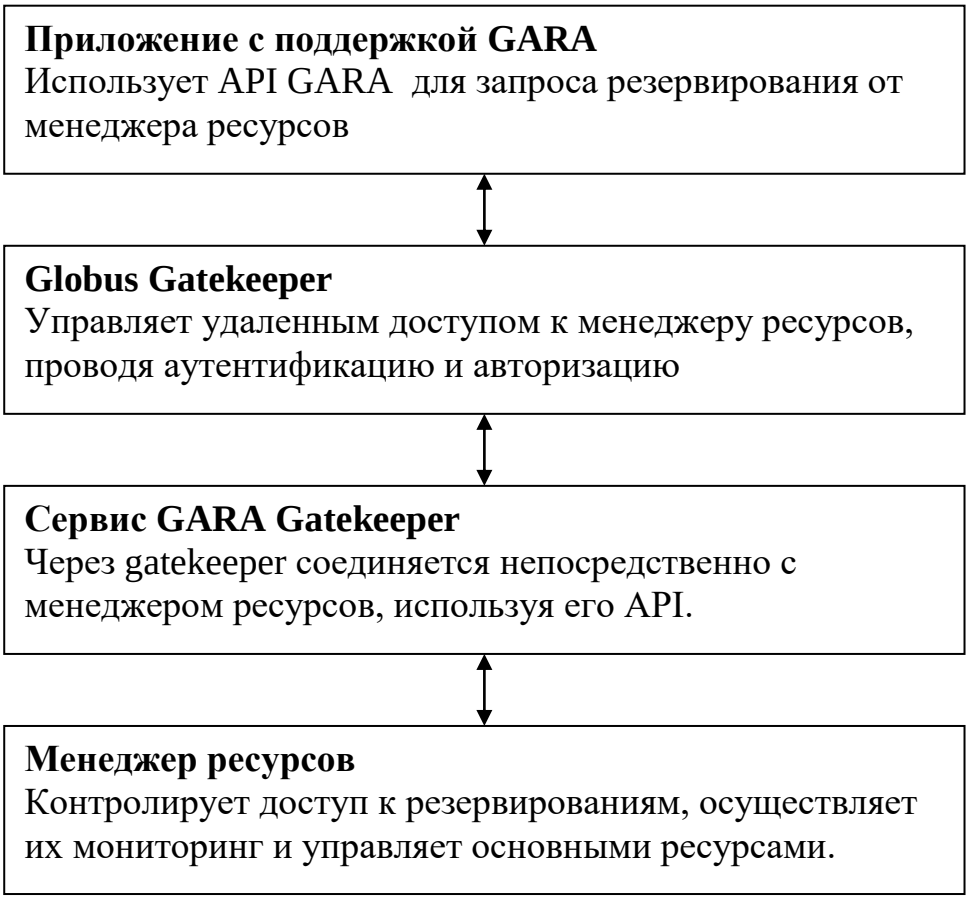

Рис. 5 Использование GARA

Рассмотрим функциональные возможности API. Резервирование в GARA производится по отдельности для каждого из типов ресурсов, что можно объяснить стремлением разработчиков предоставить базовое средство для создания резервирований. В общем случае схема резервирования следующая. Сначала приложение создает **запрос**, с указанием запрашиваемого ресурса, длительности резервирования, а также промежутка времени, в который должно попасть время старта резервирования и передает его менеджеру ресурсов. После того как запрос получен, менеджер ресурсов проверяет возможность создания такого резервирования, если возможно, создает его и посылает приложению дескриптор, однозначно определяющий созданное резервирование и включающий, в частности, способ связи с менеджером ресурсов. Все дальнейшие операции с резервированием осуществляются с помощью только этого дескриптора, без указания, например, сетевого адреса.

Для того чтобы зарезервировать набор ресурсов, надо сделать несколько резервирований, но при этом становится проблемой добиться их одновременности. Для решения этой задачи предусмотрено **двухходовое резервирование**. При таком подходе процесс создания резервирования разбивается на две части. Сначала создается запрос и отправляется менеджеру ресурсов. Если запрос удовлетворен, менеджер ресурсов создает резервирование и посылает приложению его дескриптор. Однако через некоторое время оно будет отменено. Чтобы этого не произошло, заказавшее резервирование приложение должно успеть его **зафиксировать**. Схема резервирования комплекса ресурсов выглядит так: приложение заказывает ресурсы по отдельности и, если все заказы удовлетворены, фиксирует все созданные резервирования. В противном случае приложение ничего не делает, и резервирования через некоторое время отменяются.

Еще одной интересной особенностью API GARA является возможность **связывания** резервирований. Действительно, некоторые резервирования не могут быть обработаны без дополнительной информации, которая определяется после выполнения резервирования. Поэтому, GARA позволяет создать резервирование без предоставления детализированного запроса. Однако перед использованием такого резервирования необходимо предоставить отсутствующие атрибуты. Для примера рассмотрим сетевое резервирование. Пусть нам для выполнения приложения требуются два компьютера. Для обеспечения возможности обмена сообщениями, на них необходимо выделить под это приложение какой-то канал связи. Кроме указания промежутка времени и пропускной способности мы должны, в частности, указать и порты, которые будет использовать наше приложение. Но до того как сделано резервирование, адреса портов не известны. Исходя из этого, предлагается следующая схема действий. Сначала делается резервирования на обоих компьютерах (A и B), но пока не указываются порты. Далее, получаем номера портов (P_A и P_B соответственно), которые выделяются локальными планировщиками. После этого вносим уточнения в сделанные резервирования, указывая порты (для резервирования на A - P_B, а для резервирования на B – P_A).

Запрос на резервирование имеет пять основных атрибутов:

- Start Time – Самое раннее время, в которое резервирование может начинаться.
- Duration – Продолжительность резервирования в секундах.
- Reservation Type – Тип резервируемых ресурсов.
- Reservation Subtype – Детали резервирования, например, для резервирования сети – IP точек соединения.

- Resource-Specific Parameters – Собственно описание запроса (для памяти – объем, для сети – ширина канала и т.п.).

Также может использоваться атрибут End Time – время, к которому резервирование должно закончиться.

API GARA позволяет осуществлять контроль за созданными резервированиями, изменять их, отменять. Используя дескриптор с помощью API можно связать резервирование, а также осуществить обратное действие – развязать, т.е. удалить информацию о некоторых деталях резервирования. В этом случае для его использования связывание надо будет произвести вновь. Можно получить информацию о деталях резервирования и о его состоянии. Также предусмотрена специальная функция - Register Callback, оповещающая об изменении состояния резервирования или же передающая дополнительную информацию приложению от менеджера ресурсов.

5. Выводы и заключения

В распределенном сетевом варианте многие хорошо изученные задачи системного программирования получают новую окраску. Специфика диспетчеризации в СРВ обусловлена, в частности, ее двухуровневой архитектурой: задания распределяются по ВК, которые управляются системами пакетной обработки. В таких условиях возможность создания эффективных алгоритмов глобального уровня ограничивается тем, что наиболее распространенные на сегодняшний день СПО используют простейшие механизмы планирования и располагают ограниченными внешними интерфейсами.

В этой связи вызывает интерес и представляется перспективным подход, основанный на предварительном резервировании ресурсов. Этот подход впервые реализован в планировщике локального уровня Maui. Maui, однако, изначально не был рассчитан на глобальную операционную среду, какой является СРВ, и на задачу метадиспетчеризации. Поэтому некоторые решения, которые хороши для СПО, в СРВ оказываются неудачными или избыточными. Неудивительно, что наблюдается расхождение спецификаций интерфейса Maui и протокола удаленного резервирования GARA. Если сравнить два этих варианта с точки зрения метадиспетчеризации, можно сформулировать ряд свойств протокола предварительного резервирования, которые представляются необходимыми.

1. **Резервирование должно осуществляться под конкретное задание, и зарезервированные ресурсы не должны быть доступны никакому другому заданию.**

2. **Необходимо запретить осуществление резервирования, захватывающего уже зарезервированные ресурсы.**
   Если этого не сделать, то в случае какой-либо заминки при передаче актуальных данных о состоянии всего множества резервирований некоторые ресурсы могут быть зарезервированы одновременно под два различных задания, что в свою очередь может помешать их работе. Ни Maui, ни GARA не запрещают (это не недостаток, просто такой подход позволяет формулировать более гибкие условия доступа к ресурсам) создание резервирований на уже занятые ресурсы, но Maui предоставляет средства для получения полной картины резервирований на хосте, а в GARA таковых не предусмотрено. Соответственно, Maui можно сравнительно легко дополнить средствами блокировки дублирующих резервирований.
3. **Должна быть возможность резервирования под одно задание сразу комплекса ресурсов на хосте.**
4. **Должна быть возможность использования двухфазного механизма резервирования, включающего создание резервирования и его подтверждение.**
   В случае не подтверждения резервирования в течении некоторого промежутка времени, сделанное резервирование отменяется. Возможность использования такой опции позволит избавить пользователя от необходимости отменять сделанное резервирование в том случае, если ему удалось сделать другое, более ему подходящее. В GARA этот механизм реализован, в Maui – нет.
5. **Надо предусмотреть механизм автоматической отмены резервирований после завершения того задания, под которое они сделаны.**
   Так как редкое задание выполняется ровно такое количество времени, которое содержится в параметре walltime, то после завершения задания ресурсы остаются зарезервированными еще некоторое время, что ведет к потерям. Ожидать самостоятельной отмены резервирования пользователем не представляется разумным, т.е. необходимо средство автоматизации этого процесса. Это средство должно иметь доступ и к очереди, и к таблице резервирований (расписанию). GARA не дает возможности ни того, ни другого, а Maui, по крайней мере только средствами внешнего API, не позволяет добраться до очередей.
6. **Должен быть реализован механизм "связывания" резервирования, т.е. уточнения некоторых параметров после того, как резервирование осуществлено.**
   Наличие такой возможности представляется полезным. Это может помочь в ситуациях, когда для резервирования необходимы какие-то данные, которые невозможно получить в момент его создания. Таковыми могут

быть номера портов, ID процессов, имена заданий в очереди планировщика, и т.д. "Связывание" реализовано в GARA и отсутствует в Maui.

Кроме непосредственной полезности для метадиспетчеризации предварительное резервирование имеет дополнительный потенциал, заключенный в поддерживающем механизме. Как было показано, для job-резервирования строится “расписание” обработки заданий на будущее. Посредством этого становится возможным, в принципе, предсказать время старта любого задания, если известен его ресурсный запрос.

Таким образом, в ближайшее время можно ожидать появления нового поколения систем пакетной обработки с улучшенной функциональностью и расширенным интерфейсом. Развитие средств метадиспетчеризации заданий стоит ориентировать на эти возросшие возможности.

Литература

[1]. Larry Smarr. Computational Infrastructure: Toward the 21st Century.CACM, November 1997/Vol. 40, No. 11, 29-32

[2]. I.Foster, C.Kesselman. Globus: A metacomputing infrastructure toolkit. Int. J. Supercomput. Appl., 1997.

[3]. R.Stevens, P.Woodward, T.DeFanti, C.Catlett. From the I-WAY to the National Technology Grid, CACM, 40(11):50-61, 1997

[4]. http://www.globus.org

[5]. http://www.npaci.edu/

[6]. http://www.eu-datagrid.org/

[7]. http://www.nas.nasa.gov/About/IPG/ipg.html

[8]. S. Barnard, R. Biswas, S. Saini, R.Van der Wijngaart, M. Yarrow, L. Zechter, I. Foster, O. Larsson. Large-Scale Distributed Computational Fluid Dynamics on the Information Power Grid using Globus, http://www.globus.org/documentation/incoming/paper1.pdf

[9]. Коваленко В.Н., Коваленко Е.И., Корягин Д.А, Любимский Э.З., Хухлаев Е.В. Управление заданиями в распределенной вычислительной среде. Открытые системы, № 5-6 (2001), стр. 22-28, http://www.osp.ru/os/2001/05-06/022.htm

[10]. С.А.Богданов, В.Н.Коваленко, Е.В.Хухлаев, О.Н.Шорин. “Метадиспетчер: реализация средствами метакомпьютерной системы Globus”. Препринт ИПМ РАН, № 30, 23 с., Москва, 2001

[11]. Quinn Snell, Mark Clement, David Jackson, Chad Gregory. The Performance Impact of Advance Reservation Meta-scheduling. Computer Science Department Brigham Young University Provo, Utah 84602-6576, 2000, http://supercluster.org/research/papers/ipdps2000.pdf

[12]. http://www.supercluster.org

[13]. http://pbs.mrj.com

[14]. http://www.redbooks.ibm.com/pubs/pdfs/redbooks/sg245522.pdf

[15]. K. Czajkowski, V. Sander. Grid Resource Management Protocol: Requirements. University of Southern California, September 11, 2001. http://www.cs.nwu.edu/~jms/sched-wg/WD/schedWD12.1.pdf

[16]. http://www.mcs.anl.gov/qos/