Исследование возможностей интеграции и особенностей программной реализации новых алгоритмов планирования в планировщике Maui

С.В. Минухин

Харьковский национальный экономический университет, Харьков

ms_vl@mail.ru

Аннотация. *Рассмотрены вопросы, связанные с возможностями использования новых алгоритмов планирования в планировщике Maui. Исследован и детально изучен исходный код планировщика, что позволило выделить основные этапы планирования работ и определить основные функции, которые используются в программной реализации существующего планировщика. Рассмотрены пути и варианты интеграции нового алгоритма планирования и модификации программного обеспечения планировщика Maui.*

Ключевые слова

Ресурсы, Кластер, Система управления пакетной обработки заданий, Torque, Maui, Планировщик, Наименьшее покрытие.

1 Введение

При проектировании распределенной вычислительной системы (кластера) актуальной является задача выбора оптимального алгоритма планирования выполнения работ очереди, который эффективно распределяет ресурсы (брокера ресурсов). Как правило, планировщик настраивается администратором кластера, исходя из имеющегося опыта, типов запускаемых работ, используемых политик планирования и т.д. На кластере, используемом реальными пользователями, отсутствует возможность сравнить работу планировщиков на одном наборе задач, а потому сложно применить формальный критерий для сравнения, который зависит от конкретного набора задач. Качество предоставляемого сервиса в сфере высокопроизводительных вычислений определяется эффективностью функционирования брокера ресурсов: согласованностью распределения имеющихся ресурсов, их высокой загрузкой, минимальным временем ожидания работ очереди, быстрое планирование вычислительных процессов.

Вопросы повышения эффективности работы брокеров ресурсов распределенных вычислительных систем рассматривались в работах [1, 2], где предложены методы повышения эффективности работы брокера ресурсов Nordugrid ARC 2.0, в работе [3] рассмотрены возможные подходы к интеграции новых алгоритмов планирования в планировщик Maui.

Целью данного исследования является развитие подходов, предложенных в [3], детальный анализ особенностей работы планировщика Maui, а также определение тех программных модулей, в которые необходимо внести изменения для интеграции новых методов планирования в программное обеспечение планировщика Maui для повышения эффективности процессов планирования.

2 Анализ процессов и механизмов планирования ресурсов кластера под управлением планировщика Maui

Как и другие планировщики, Maui работает итеративно, т.е. перемежая процесс планирования с ожиданием или выполнением внешних команд. Каждая итерация планирования начинается при осуществлении одного из следующих событий:

изменяется состояние работы или ресурса,
достигнуто границу резервирования,
получена внешняя команда,

с начала предыдущего цикла прошло время, определенное, как максимальное

Понятие резервирования формализуется в Maui соответствующим типом объектов.

Объект «резервирование» состоит из:

• ACL (Access Control List, список контроля доступа) – набора параметров, руководствуясь которым планировщик определяет, для каких работ доступны зарезервированные ресурсы;

• списка зарезервированных ресурсов;

• времени действия резервирования.

На уровне реализации каждое резервирование физически представляет собой запись в базе данных Maui. Объект резервирования привязан, с одной стороны, к времени, причем к будущему времени, с другой – интерпретация резервирования предполагает рассмотрение состояний ресурсов на моменты времени начала резервирования. Изображается это в виде временной развертки (time line) состояний ресурсов: для каждого ресурса на оси времени относятся метки, определяющие, какие ресурсы для работы в данный момент могут быть доступны. Совокупность определенного количества меток и образует одно резервирование (так зарезервировать можно целый комплекс ресурсов, и все они будут принадлежать одному резервированию) [4].

Заметим, что предварительный заказ производится под работу, которой еще нет в СПО. Таким образом, должен быть реализован такой механизм, который позволяет связать работы с проведенным резервированием. Объект резервирования имеет список контроля доступа ACL, и именно в соответствии с ним происходит привязка работ к определенному резервированию. Зарезервированные ресурсы могут получить только такие работы, которые имеют хотя бы один из параметров: GROUPLIST, USERLIST, ACCOUNTLIST, CLASSLIST и QOSLIST, что совпадает по значению с соответствующими атрибутами ACL резервирования. Поэтому привязка может осуществляться по регистрационному имени пользователя, его группы, класса и качества обслуживания (QoS). Возможности настройки списков доступа ACL в Maui достаточно большие и, в частности, можно добиться того, чтобы зарезервированные ресурсы были доступны только для определенной работы, причем даже еще не поступившей в очередь СПО (именно это нужно для метадиспетчеризации) [9].

Maui позволяет зарезервировать 4 наиболее важных типа ресурсов:

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

Количество ресурсов определяется конкретными работами. В одну работу может входить несколько параллельных заданий, каждое из которых выполняется на одном узле. Не следует путать работы и задания (в документации Maui используются, соответственно, job и task). Каждая работа может состоять из нескольких заданий. Задание может выполняться только на одном узле и является, как бы, квантом работы. Когда осуществляется резервирование под работу, то в запросе указываются ресурсы, необходимые для одной работы и количество заданий. Таким образом, все работы выдаются однотипными, по крайней мере, которые требуют одинаковое количество ресурсов. Далее будем рассматривать те работы, которые имеют только одно задание.

В Maui предусмотрены средства для операций с существующими резервированиями. С помощью команды showres можно получить информацию о том, где, для каких задач, на какое время и сколько ресурсов зарезервировано, причем доступен поиск, как по резервированию, так и по узлам. Существует также команда releaseres отмены созданных резервирований[4].

В процессе планирования применяется еще один, внутренний тип резервирования Maui - job-резервирования. С точки зрения реализации он мало отличается от описанного выше административного резервирования.

Оно используется в двух случаях:

• для обеспечения доступности ресурсов для выполняемой работы. Как только работа запускается, то сразу, на все время выполнения (walltime), создается job-резервирование, что закрывает доступ к ресурсам, необходимым для этой работы, всем, кроме него самого. Может случиться, что во время работы все эти ресурсы и не будут использоваться, но как только они потребуются работе, оно гарантированно сможет их получить. Когда работа завершается (возможно, раньше, чем через время walltime), job-резервирование отменяется, и ресурсы становятся свободными [4];

• job-резервирование применяется для того, чтобы работы с более высокими приоритетами не были задержаны запуском менее приоритетных работ, что может случиться при работе алгоритма Backfill. В общем виде процесс запуска работ выглядит так: до тех пор, пока это возможно, упорядоченные по приоритетам работы берутся из очереди и запускаются (попутно создается job-резервирование), как только очередную работу нельзя запустить из-за отсутствия ресурсов, для него определяется в ближайшее время, в которое можно будет произвести запуск, и начиная с него создается job-резервирование требуемых ресурсов на время walltime.

После того, как сделано некоторое сконфигурированное количество резервирований, начинается работа алгоритма Backfill. Он выясняет, какие узлы и на какое время, начиная с текущего, свободны (это определяется сделанным job-резервированием). После этого свободные узлы объединяются в «окна». Суммарная ширина

«окон» может оказаться больше количества свободных на данный момент узлов, так как некоторые узлы могут входить в несколько окон. Затем из всех «окон» выбирается одно, как правило, самое широкое. Среди всех прочих работ выбирается и запускается работа, наиболее точно удовлетворяющая этому «окну». Так как при запуске работ алгоритмом Backfill учитываются сделанные job-резервирования, то соответствующие им работы не будут задерживаться [4].

Основным отличием job-резервирования от административного является то, что оно осуществляется самим планировщиком и им же отменяется перед началом очередного шага планирования. При этом состояние резервирования может отличаться от предыдущего шага планирования весьма существенно. Это имеет место в случае, когда поступают новые работы, изменяющие порядок в очереди или освобождающие ресурсы. В отличие от административных резервирований, job-резервирование не гарантирует запуск работы в указанное время. Таким образом, job-резервирование (не для запущенных работ) является всего лишь метками, позволяющими корректно учесть в алгоритме Backfill приоритеты работ. Административные и job-резервирования существуют параллельно, т.е. job-резервирования могут накладываться на административные, если обычная работа, под которую делается job-резервирование, удовлетворяет ACL административной работы [5].

Рассмотрим планирование на основе приоритетов, которое предусматривает, что ресурсы для более приоритетных работ выделяются раньше, чем для менее приоритетных. В этих условиях широко используются алгоритмы типа FCFS (FIFO), которые работают по принципу выделения свободных ресурсов самой приоритетной работе из очереди, которая может на них разместиться (решиться). Этот тип планирования использует базовый планировщик TORQUE (демон pbs_sched). При этом получается, что большая часть узлов будет всегда занята мелкими работами, т.е. возникает фрагментация ресурсов. Даже, если работа имеет наивысший приоритет, необходимый ей объем ресурсов может никогда не образоваться и, следовательно, работа может никогда не начать выполняться. Для среды, обслуживающей однопроцессорные работы, этой проблемы не существует. Она возникает в случае, когда есть разделяемые ресурсы, например, при обслуживании многопроцессорных работ. Аналогичная ситуация возникает на ресурсах с общей памятью, в среде с общим файловым пространством и других случаях, когда ресурсы делятся между работами, а не выделяются под работу полностью [6,7].

Для решения проблемы коаллокации в планировщике Maui используется алгоритм обратного заполнения Backfill [8], разработанный для использования в многопроцессорных системах (MPP) типа IBM SP2.

Он имеет следующие преимущества:

в условиях работы в приоритетной системе позволяет избежать зависания работы, гарантируя его запуск;

эффективно загружает ресурсы, позволяя избежать их фрагментации;

имеет приемлемые временные характеристики при работе на большом количестве вычислительных узлов;

позволяет работать на множестве гетерогенных ресурсов.

Алгоритм обратного заполнения Backfill работает по следующему принципу: размещая наиболее приоритетную работу, он определяет момент времени, когда освободится достаточное количество ресурсов, занятых работами, которые на настоящий момент времени выполняются, и выполняет резервирование этих ресурсов. Работа с меньшим приоритетом может быть запущена вне очереди, но только в том случае, если оно не будет мешать запуску более приоритетных работ (в консервативном варианте Backfill). Отметим, что в опубликованных работах по алгоритму Backfill содержится лишь упомянутый выше принцип, а также общая схема процесса распределения работ очереди по ресурсам.

Отметим, что алгоритм Backfill имеет свои недостатки, главным из которых является то, что когда все работы уже распределены на ресурсы кластера и выполняются, новые работы, которые поступают в очередь на кластер, не распределяются равномерно на ресурсы кластера (на первый ресурс, которой освободился), а попадают на первый ресурс из всего списка ресурсов, т.е. в начало этого списка. Этот недостаток приводит к уменьшению производительности и быстродействия выполнения работ. Возможным решением этой проблемы может быть доработка планировщика Maui путем внедрения в него алгоритмов планирования, позволяющих устранить этот недостаток[9].

3 Суть нового алгоритма планирования и особенности его интеграции в состав планировщика Maui

Исходный код планировщика Maui находится в свободном доступе на ресурсе www.adaptivecomputing.com. Проведенный анализ дистрибутива планировщика Maui позволяет сделать вывод о том, что он написан на языке программирования С. Исходный код планировщика содержит примерно 160000 строк кода и базируется на планировщике Moab, который в настоящее время является коммерческим продуктом. Было выявлено, что основным алгоритмом планирования в Maui (Moab) является Backfill. Так же можно сделать вывод о том, что Maui (Moab) и является его реализацией со вспомогательными механизмами планирования, такими, как политики справедливого распределения ресурсов и др. Maui обладает всеми возможностями планировщика Moab по состоянию на 2003 г., т. е. за все функции управления процессом планирования отвечает исходный код

Moab. Кроме этого, в процессе анализа исходного кода планировщика Maui были выявлены попытки его модификации другими организациями путем подключения новых алгоритмов выделения ресурсов и политик справедливого распределения ресурсов. Таким образом, можно сделать вывод о том, что интеграция нового алгоритма планирования ресурсов возможна и является достаточно актуальной задачей.

Основным алгоритмом планирования в Maui является BackFill (алгоритм обратного заполнения), но, как и любой другой метод, он имеет свои недостатки. Учитывая то, что современные кластерные системы имеют определенную гетерогенность, представляется целесообразным интегрировать в планировщик ресурсов Maui новый алгоритм планирования, который будет более эффективным в условиях гетерогенности ресурсов кластера и высокой интенсивности поступления работ в систему.

Основная идея, положенная в основу нового алгоритма, состоит в использовании в качестве процедуры планирования решения задачи о наименьшем покрытия (ЗНП).

Суть метода ЗНП заключается в том, чтобы спланировать решение как можно больше работ, привлекая при этом как можно меньше ресурсов.

Входными данными для решения ЗНП является матрица соответствия. Эта матрица отражает данные о том, какие задания на любом ресурсе могут быть решены с учетом, что на одном ресурсе может выполняться несколько заданий (рис. 1) [10].

Задания	Ресурсы							
	R_{1*}	R_2	R_{3*}	R_4	R_{5*}	R_6	R_7	R_8
$task_1$	1							
$task_2$		1	1	1				
$task_3$	1					1		
$task_4$			1					1
$task_5$		1	1	1				
$task_6$	1						1	
$task_7$					1			1
$task_8$					1		1	
$task_9$					1			
$task_{10}$		1	1					1
$task_{11}$		1			1			
$task_{12}$	1			1		1		

Рис. 1. Матриця соответствия «задания–ресурсы»

Для построения этой матрицы предлагается программно использовать функции из исходного кода планировщика Maui. Планировщик является очень сложной системой с множеством параметров и факторов, которые учитываются при планировании, также в него достаточно плотно интегрирован алгоритм планирования Backfill, что весьма затрудняет процесс интеграции нового алгоритма. В связи с этим, чтобы не вносить эвристику в работу планировщика, предлагается выполнять процедуру так называемого «матчинга» (matching) – определения того, на каком из ресурсов может выполняться то или иное задание – функциями самого планировщика, а в качестве параметров передавать ей работы, имеющиеся ресурсы, и требования на ресурсы.

После формирования этой матрицы (рис. 1) решается ЗНП. В процессе реализации алгоритма планирования предполагается разработка алгоритма обработки исключений, т.е. обработки запросов, которые по той или иной причине не прошли процедуру планирования. Результатом решения ЗНП является вектор назначений, т.е. определения множества работ, которое будет решаться на различных ресурсах.

Отладка планировщика Maui осуществляется в интегрированной среде разработки Netbeans и Eclipse. Факт того, что планировщик Maui очень тесно интегрирован с алгоритмом планирования Backfill, существенно усложняет процесс интеграции нового алгоритма. Изменения в одном фрагменте исходного кода ведут к значительным проблемам и ошибок во всем коде.

Поэтому одной из проблем, которая возникла в процессе изучения работы планировщика и интеграции нового алгоритма в состав планировщика Maui, является недостаточный опыт разработки программного обеспечения под управлением Unix-систем.

На сегодняшнее время новый алгоритм планирования реализован, но в виде Windows-приложения. Это Windows-приложение моделирует работу кластера и процессы планирования в нем. Оно позволяет провести анализ эффективности работы алгоритма при различных условиях, а именно, для различных

производительностей ресурсов, различных по сложности выполняемых работ и различной интенсивности работ входного потока.

Так как алгоритм уже программно реализован, было принято решение пойти по пути портирования программного обеспечения (исходного кода) алгоритма на платформу Unix.

Модель работы кластера имеет графический интерфейс для взаимодействия с пользователем. Первой проблемой, с которой пришлось столкнуться при интеграции, является выделение исходного кода алгоритма из Windows-приложения. Сложность решения этой проблемы заключается в том, что элементы графического интерфейса этого приложения тесно связаны с кодом алгоритма, поэтому при компиляции исходного кода алгоритма под платформой Unix возникает большое количество ошибок. С помощью этих элементов пользователю выводится разнообразная информация о процессе планирования. В процессе портирования эта проблема была решена путем модификации кода алгоритма и удаления из него элементов графического интерфейса Windows-приложения.

При реализации этого алгоритма в Windows-приложении была использована стандартная библиотека шаблонов (Standard Template Library; STL) – библиотека для языка программирования C + +, содержащая набор согласованных обобщенных алгоритмов, контейнеров, средств доступа к содержимому и различных вспомогательных функций. Процесс планирования в модели был реализован именно с помощью динамических контейнеров STL, а именно таких, как vector, set и т.д.. В них сохраняются все элементы процесса планирования. Эти контейнеры имеют собственные функции для добавления, удаления, и доступа к элементам контейнера. Проанализировав исходный код планировщика Maui, стало ясно, что в нем все элементы, которыми оперирует планировщик в процессе планирования, сохраняются в обычных статических массивах. Другой проблемой является сведение массивов и контейнеров, которая была решена путем реализации функций добавления и удаления элементов из статических массивов в динамических контейнерах и наоборот.

После анализа исходного кода планировщика стало известно, что работы и узлы кластера описываются в нем так называемыми пользовательскими типами данных (структурами). Учитывая, что Windows-приложение, которое моделирует работу кластера, представляет собой модель, то некоторые параметры работы кластера, работ и процесса планирования задаются простыми типами данных и необходимо свести эти различные типы данных. Эта задача стала третьей проблемой при интеграции нового алгоритма в состав планировщика. Все пользовательские типы данных (структуры) строятся на простых типах (рис. 2).

Учитывая это, предлагается получать доступ к нужным полям структуры, описываемым простыми типами данных при помощи операции разыменования.

За политику выделения ресурсов в файле планировщика Maui отвечает параметр NODEALLOCATIONPOLICY (рис. 3). Одним из значений этого параметра может быть значение LOCAL. Это значение позволяет вызвать локально созданный алгоритм выделения ресурсов на работы в каталоге дистрибутива maui-3. *. * \ Contrib \ nodeallocation \ (рис. 4). Именно в этом каталоге и будет находиться файл исходного кода с новым алгоритмом планирования и выделения ресурсов. После успешного завершения написания кода алгоритма файл с входным кодом будет подключен к файлу MLocal.c в каталоге maui-3. *. * \ Src \ moab \, в этом же файле будет вставлен код вызова алгоритма, а именно в функции MLocalQueueScheduleIJobs.

Вызов нового алгоритма необходимо делать в той части кода, где разработчики оставили комментарий: «NOTE: insert call to scheduling algorithm here» (рис. 5) [9].

Файл исходного кода планировщика Maui MLocal.c отвечает за работу локально созданных алгоритмов планирования и политик выделения ресурсов под работы.

В результате более детального изучения алгоритмов и механизмов планирования Maui стало ясно, что, если пойти по этому пути интеграции нового алгоритма, то большинство всех механизмов планирования Maui, таких, как проверка политик планирования, различных ограничений, резервирования и т.д. Они будут выполняться только в том случае, если разработчик модифицирует свой новый алгоритм, используя функции исходного кода самого Maui, а также, если разработчик реализует собственную проверку политик планирования и других механизмов Maui. На наш взгляд это является не самым лучшим решением, так как вряд ли механизмы, реализованные сторонними разработчиками, будут лучше работать в процессе планирования, чем те механизмы, которые уже реализованы разработчиками Maui.

Таким образом, после очередной детализации в исследовании механизмов и процессов планирования Maui было обнаружено, что за выделение ресурсов отвечает функция MJobAllocMNL, которая находится в файле MSched.c, в каталоге maui-3. *. * \ Src \ moab \ (рис. 6).

```
typedef struct mjob_t {
  char   Name[MAX_MNAME + 1]; /* job ID                                  */
  char  *AName;               /* alternate name (user specified)         */
  char  *RMJID;               /* resource manager job ID                 */
  int    Index;               /* job table index                         */
  long   CTime;               /* creation time (first RM report)         */
  long   MTime;               /* modification time (any source)          */
  long   ATime;               /* access time (most recent RM report)     */
  long   StateMTime;
  long   SWallTime;           /* duration job was suspended              */
  long   AWallTime;           /* duration job was executing              */
  mjckpt_t *Ckpt;             /* checkpoint structure                    */
  mjobcache_t C;
  macl_t      RCL[MAX_MACL]; /* required CL                              */
  mcred_t     Cred;
  mqos_t     *QReq;
  mres_t     *R;             /* reservation                             */
  char        ResName[MAX_MNAME];
  char        RAList[MMAX_JOBRA][MAX_MNAME]; /* reservation access list  */
  mrm_t      *RM;
  int         SessionID;
  char     *NeedNodes;
  msdata_t *SIData;         /* stage in data  */
  msdata_t *SOData;         /* stage out data */
  mreq_t *Req[MAX_MREQ_PER_JOB + 1];
  mreq_t *GReq;
  mreq_t *MReq;             /* primary req */
  mjobe_t E;                /* job execution environment               */
  int    NodeCount;         /* nodes assigned                          */
  int    TaskCount;         /* tasks assigned                          */
  int    TasksRequested;    /* tasks requested                         */
  int    NodesRequested;    /* number of nodes requested               */
  mnalloc_t *NodeList;    /* list of allocated hosts */
```

Рис. 2. Часть структуры данных mjob_t, описывающая работу

```
SERVERHOST            Localhost
ADMIN1                root
RMCFG[S200]           TYPE=PBS
AMCFG[bank]           TYPE=NONE
RMPOLLINTERVAL        00:00:30
SERVERPORT            42559
SERVERMODE            NORMAL
LOGFILE               maui.log
LOGFILEMAXSIZE        10000000
LOGLEVEL              3
QUEUETIMEWEIGHT       1
BACKFILLPOLICY        FIRSTFIT
RESERVATIONPOLICY     CURRENTHIGHEST
NODEALLOCATIONPOLICY  MINRESOURCE
```

Рис. 3. Пример возможного листинга файла конфигурации Maui

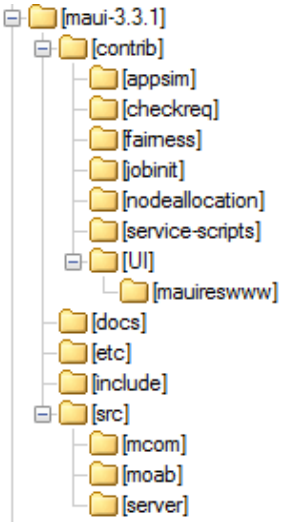

Рис. 4. Структура дистрибутива «Maui»

```
/* #include "../../contrib/sched/XXX.c" */
int MLocalQueueScheduleIJobs(
  int    *Q,
  mpar_t *P)
  {
  mjob_t *J;
  int     jindex;
  if ((Q == NULL) || (P == NULL))
    {
    return(FAILURE);
    }
  /* NOTE:  insert call to scheduling algorithm here */
  for (jindex = 0;Q[jindex] != -1;jindex++)
    {
    J = MJob[Q[jindex]];
    /* NYI */
    DBG(7,fSCHED) DPrint("INFO:     checking job '%s'\n",
      J->Name);
    }  /* END for (jindex) */
    return(FAILURE);
  }  /* END MLocalQueueScheduleIJobs() */
```

Рис. 5. Функция MLocalQueueScheduleIJobs

```
int MJobAllocMNL(
  mjob_t      *J,             /* I: job requesting resources */
  mnodelist_t  MFeasibleList, /* I: feasible nodes           */
  char        *NodeMap,
  mnodelist_t  MOutList,      /* O: allocated nodes          */
  int          NAPolicy,      /* I: node allocation policy   */
  long         StartTime)     /* I: time job must start      */
```

Рис. 6. Функция MJobAllocMNL

Именно в этой функции необходимо внести изменения и реализовать функции нового алгоритма. Для этого требуется перехватить переменную MFeasibleNodes (массив узлов, которые могут выполнять работы), обработать ее при помощи нового алгоритма и передать переменной MOutList (выделенные узлы для решения работы) список тех узлов, на которых будет выполняться работа.

В связи с предлагаемыми изменениями при интеграции нового алгоритма, появилась еще одна проблема, суть которой заключается в том, что Maui построен так, что он прогоняет каждую работу по всему циклу планирования, то есть, каждая работа проходит каждый этап и механизм планирования отдельно, и получает назначение, на каких ресурсах оно будет выполняться. В свою очередь, наш алгоритм является пакетным, т.е. обрабатывает сразу пакет работ и дает назначение работе из пакета на тот ресурс, на котором оно будет решаться.

На каждой итерации планирования все работы, которые поступили в Maui, проходят полный цикл планирования, причем так, что каждая работа после цикла планирования запускается отдельно. Итак, еще одним вариантом является отключение функции старта (MJobStart) для каждой работы и подключение ее тогда, когда все работы на одной итерации планирования пройдут полный цикл планирования. После прохождения цикла планирования каждая работа получает назначение на ресурс. Для того, чтобы не вносить эвристику в новый алгоритм, предлагается отменить это назначение и собрать все «распланированные» работы в массив и передать его новому алгоритму, после обработки которым нужно подключить функцию старта работы, а в качестве параметра передать этой функции не отдельную работу, а массив работ с назначениями на ресурсы (каждая работа в структуре ее описания имеет поле NodeList (рис. 2)), на которых они будут выполняться. Прохождение цикла планирования для каждой работы обязательно, так как в цикле работа не только получает назначение, но и проходит различные проверки политик планирования и ограничений.

Компиляция исходного кода планировщика с интегрированным алгоритмом под платформу Unix выполняется компилятором g++, который предназначен для языка программирования C ++, так как исходный код алгоритма имеет элементы STL, которые компилятор gcc для языка программирования C не поддерживает.

4 Выводы

Таким образом, сформулированы общие проблемы, связанные с имеющимися в программном обеспечении планировщика Maui возможностями интеграции в него новых алгоритмов планирования. Сложность решаемых при этом задач заключается в том, что сам исходный код не закомментирован в необходимой для его модификации мере с целью интеграции новых алгоритмов. В связи с этим были детально изучены принципы и механизмы работы планировщика, детально исследован его исходный код, определены все ключевые этапы планирования, которые проходят работы, а так же определены основные функции (MQueueBackfill, MLocalQueueScheduleIJobs, MJobSelectMNL, MJobAllocMNL, MJobStart), использование и модификация которых позволит интегрировать новый алгоритм и модифицировать программное обеспечение Maui.

Рассмотрены пути и вариантов по интеграции нового алгоритма, и проблемы, которые встречались на каждом из этих путей и возможные варианты их решения.

Литература

[1] А. Петренко. Гібридний Алгоритм брокера для Nordugrid ARC 2.0. / А.І. Петренко, С.Я. Свистунов, П.В. Свірін. // HPC UA 2012: Друга міжнародна конференція «Високопродуктивні обчислення», 8–10 жовтня, 2012, Київ, Україна 8-10 жовтня : матеріали. – К. : Національна академія наук України, 2012. – С. 275–277.

[2] Свірін П.В. Методика вдосконалення брокера для NORDUGRID ARC. // Вісник НТУУ «КПІ»: Інформатика, управління та обчислювальна техніка. Зб. наук. праць. – 2012. – Вип. 57. – С. 31–35.

[3] Минухин С.В. Метод планирования ресурсов в распределенной гетерогенной системе и исследование его практической реализации. / С.В. Минухин, С.В. Баранник, С.В. Знахур, Р.И. Зубатюк, А.В. Соболев. // HPC UA 2012: Друга міжнародна конференція «Високопродуктивні обчислення», 8–10 жовтня, 2012, Київ, Україна 8-10 жовтня : матеріали. – К. : Національна академія наук України, 2012. – С. 247–254.

[4] Adaptive Computing – Documentation / Maui Administrator's Guide. Version 3.2 [Електронный ресурс]. – Режим доступа к ресурсу: http://www.adaptivecomputing.com/resources/docs/maui/index.php.

[5] Мінухін С.В. Дослідження методів локальних планувальників ресурсів та їх модифікації в ГРІД-системах./С.В. Мінухін, О.В. Мєзєнцев. // Системи обробки інформації. – 2012. – Вип. 4 (102). Т.1. – С. 42–48.

[6] Система пакетной обработки заданий Torque: Руководство пользователя. Т – Платформы, 2008. [Електронный ресурс]. – Режим доступа к ресурсу: hpc.ssau.ru/files/doc/torque_manual.pdf.

[7] Adaptive Computing – Documentation / TORQUE Administrator's Guide. Version 3.0.3 [Електронний ресурс]. – Режим доступа к ресурсу: http://www.adaptivecomputing.com/resources/docs/torque/3-0-3/index.php.

[8] Коваленко В.Н. Использование алгоритма BACKFILL в ГРИД / В.Н. Коваленко., Д.А. Семячкин.. [Електронний ресурс]. – Режим доступа к ресурсу: http://gridclub.ru/library/publication.2004-12-27.6965428621/publ_file/.

[9] Коваленко В.Н. Полигон Грид в ИПМ РАН и разработка методов управления ресурсами в глобальной среде / В.Н. Коваленко, Д.А. Корягин. // X конференция представителей региональных научно-образовательных сетей, RELARN-2003, Санкт-Петербург, 16-20 июня 2003.// Сборник тезисов докладов, Тровант, 2003. – С. 214–216.

[10] Листровой С.В. Модель и подход к планированию распределения ресурсов в гетерогенных Грид-системах /С.В. Листровой, С.В. Минухин.// Международный научно-технический журнал «Проблемы управления и информатики». – 2012. – № 5. – С. 120–133./