УДК 519.687

СИМУЛЯТОР ВЫЧИСЛИТЕЛЬНОГО КЛАСТЕРА И ЕГО УПРАВЛЯЮЩЕЙ СИСТЕМЫ, ИСПОЛЬЗУЕМЫЙ ДЛЯ ИССЛЕДОВАНИЯ АЛГОРИТМОВ ПЛАНИРОВАНИЯ ЗАДАЧ

П.Н. Полежаев

SIMULATOR OF COMPUTER CLUSTER AND ITS MANAGEMENT SYSTEM USED FOR RESEARCH OF JOB SCHEDULING ALGORITHMS

P.N. Polezhaev

В данной работе описывается симулятор вычислительного кластера и его управляющей системы, учитывающий его топологию, коммуникационные задержки при передаче данных, а также многопроцессорность вычислительных узлов. Он применяется для исследования алгоритмов планирования параллельных задач на вычислительном кластере. Разработана имитационная схема и модель кластера, модель его вычислительной загрузки, приводятся исследуемые алгоритмы планирования, а также описывается система критериев и метрик их сравнения.

Ключевые слова: симулятор вычислительного кластера и его управляющей системы, планирование задач, высокопроизводительные вычисления, имитационное моделирование

In this work the simulator of computer cluster and its management system which considers its topology, communication latency in data transmition and multiprocessor nodes is described. It is used for investigation of parallel jobs scheduling algorithms on computer cluster. The simulating scheme and model of cluster, its computational load model were developed, job scheduling algorithms were presented, and also the system of criteria and metrics for their comparison is described.

Keywords: a simulator of computer cluster and its management system, jobs scheduling, high performance computings, a simulation modeling

Введение

В настоящее время весьма актуальной является проблема эффективного планирования параллельных задач, поступающих в очередь управляющей системы вычислительного кластера, т.к. современные алгоритмы планирования не обеспечивают должного качества составляемых расписаний запуска задач. Весьма часто возникает ситуация, при которой в расписании остается значительное число окон, в то время, когда в очереди имеется большое количество ожидающих своего исполнения задач. Это приводит к снижению производительности всей вычислительной системы.

Другим существенным недостатком современных управляющих систем вычислительного кластера является использование алгоритмов планирования, не учитывающих топологию при размещении процессов задач на вычислительных узлах, а также их многопроцессорность. Учет данных факторов позволит увеличить эффективность работы алгоритмов планирования.

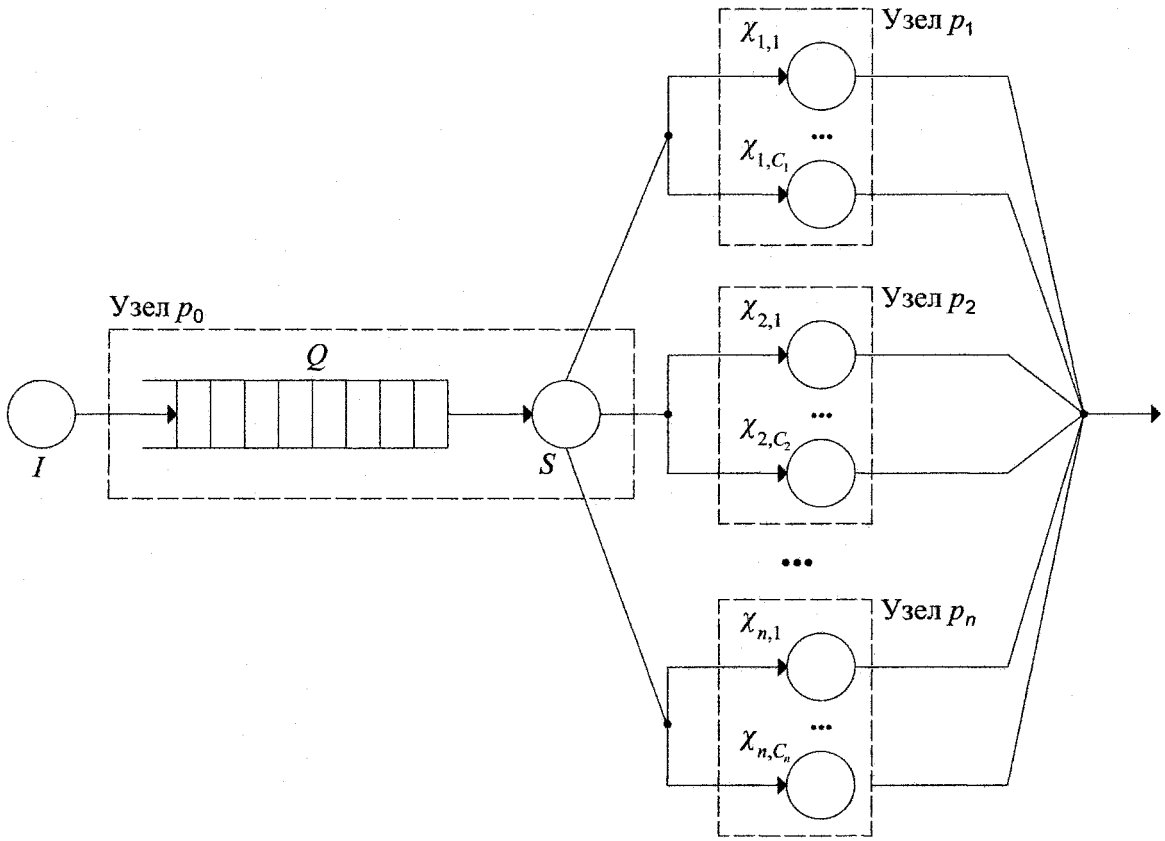

Рис. 1. Имитационная схема управляющей системы вычислительного кластера

В связи с этим возникает необходимость исследования существующих алгоритмов планирования и разработка их новых более эффективных вариантов. Для этой цели в настоящей работе создается симулятор вычислительного кластера и его управляющей системы, учитывающий ее топологию, коммуникационные задержки при передаче данных, а также многопроцессорность вычислительных узлов.

Использование симулятора для исследования алгоритмов планирования перед применением реальной управляющей системы, исполняющейся на настоящем кластере, имеет ряд преимуществ. Во-первых, имеется возможность обеспечения контролируемой вычислительной среды – не надо учитывать возможные проблемы, связанные с необходимостью обеспечения отказоустойчивости или безопасности. Во-вторых, при использовании симулятора обеспечивается гибкость и простота модификации аппаратной конфигурации и режимов работы вычислительного кластера. И, в-третьих, в случае применения реального кластера и управляющей системы имела бы место проблема формирования потоков реальных параллельных задач, отвечающих заданным характеристикам, что, напротив, делается очень просто при использовании имитационного моделирования.

1. Имитационная схема и модель вычислительного кластера

Имитационная схема управляющей системы вычислительного кластера приведена на рис. 1. Источник I генерирует поток параллельных задач, отправляемых пользователями в очередь Q управляющего узла p_0. Канал S представляет собой планировщик, который в

соответствии с заложенным в него алгоритмом осуществляет извлечение задач из очереди Q и назначение их на свободные вычислительные ядра подходящих по конфигурации узлов.

Множество $P = \{p_1, p_2, \ldots, p_n\}$ содержит все вычислительные узлы кластера, связанные с выделенным узлом p_0 с помощью отдельной управляющей сети. Пусть $\mathrm{X}_i = \{\chi_{i,1}, \chi_{i,2}, \ldots, \chi_{i,C_i}\}$ – множество вычислительных ядер узла p_i. Они могут относиться как к отдельным процессорам узла (по одному ядру на каждом), так и к нескольким многоядерным процессорам. С целью упрощения модели они рассматриваются, как единое множество X_i. Обозначим $\mathrm{X} = \bigcup_{i=1}^{n} \mathrm{X}_i$ – множество вычислительных ядер всех узлов кластера.

Конфигурация узлов вычислительного кластера и соединяющей их высокопроизводительной сети может быть описана с помощью следующего взвешенного ориентированного графа:

$$G_T = (P, K, E, b, c, m, d, s), \tag{1}$$

где $K = \{k_1, k_2, \ldots, k_z\}$ – множество коммутаторов, E – множество направленных сетевых связей между ними, $b\colon E \to \mathbb{Z}_+ \bigcup \{0\}$ – отображение, характеризующее пропускную способность каждой сетевой связи в байтах в секунду. Функции $c, m, d\colon P \to \mathbb{Z}_+$ определяют для каждого вычислительного узла p_i соответственно количество его вычислительных ядер $c(p_i) = C_i$, объемы оперативной $m(p_i) = M_i$ и дисковой памяти $d(p_i) = D_i$ в килобайтах. Отображение $s\colon P \to \mathbb{R}_+$ для узла p_i задает относительную производительность $s(p_i) = S_i$ каждого его вычислительного ядра, которая определяет, во сколько раз ядра данного узла работают быстрее вычислительных ядер самого непроизводительного узла кластера.

Заметим, что орграф (1) позволяет моделировать статические и динамические сети с полнодуплексными и полудуплексными связями. Настоящая модель также позволяет рассматривать однородные и неоднородные вычислительные системы, как по составу вычислительных узлов, так и по конфигурации высокопроизводительной сети. В рамках проводимого исследования будут рассматриваться только однородные вычислительные сети с полнодуплексными связями.

Статические параметры C_i, M_i, D_i и S_i описывают конфигурацию вычислительного узла p_i, а динамические параметры характеризуют его состояние в произвольный момент времени $t \in [0; +\infty)$: $u_i(t)$ – загруженность вычислительных ядер, $m_i(t)$ и $d_i(t)$ – объемы доступной оперативной и дисковой памяти.

Имитационная схема вычислительного узла p_i приведена на рис. 2. p_i соединен с другими узлами и коммутаторами вычислительной сети с помощью r_i дуплексных связей. Все входящие пакеты, а также пакеты сообщений, генерируемых процессами, выполняющимися на вычислительных ядрах, сначала поступают в очередь $Q_{inp.i}$, а затем маршрутизируются каналом обслуживания R_i. Если соответствующий пакет предназначен для локального вычислительного ядра, то он передается ему непосредственно, иначе – помещается в одну из очередей $Q_{out.i,1}, Q_{out.i,2}, \ldots, Q_{out.i,r_i}$, соответствующую выбранной алгоритмом маршрутизации исходящей связи. Предполагается использование алгоритмов статической и динамической маршрутизации, характерных для конкретных топологий вычислительной системы, в том числе алгоритмы кратчайших коммуникационных путей и покоординатной маршрутизации.

При выполнении маршрутизации каналом R_i моделируется временная задержка величиной $l_{routing}$. Заметим, если два процесса одной задачи выполняются на соседних ядрах узла, то обмен пакетами между ними также осуществляется через очередь $Q_{inp.i}$ и канал R_i.

Имитационная схема коммутатора, приведенная на рис. 3, представляет собой урезанный вариант имитационной схемы узла. Отличие только в том, что коммутатор не может выполнять вычисления, и поэтому у него отсутствуют вычислительные ядра.

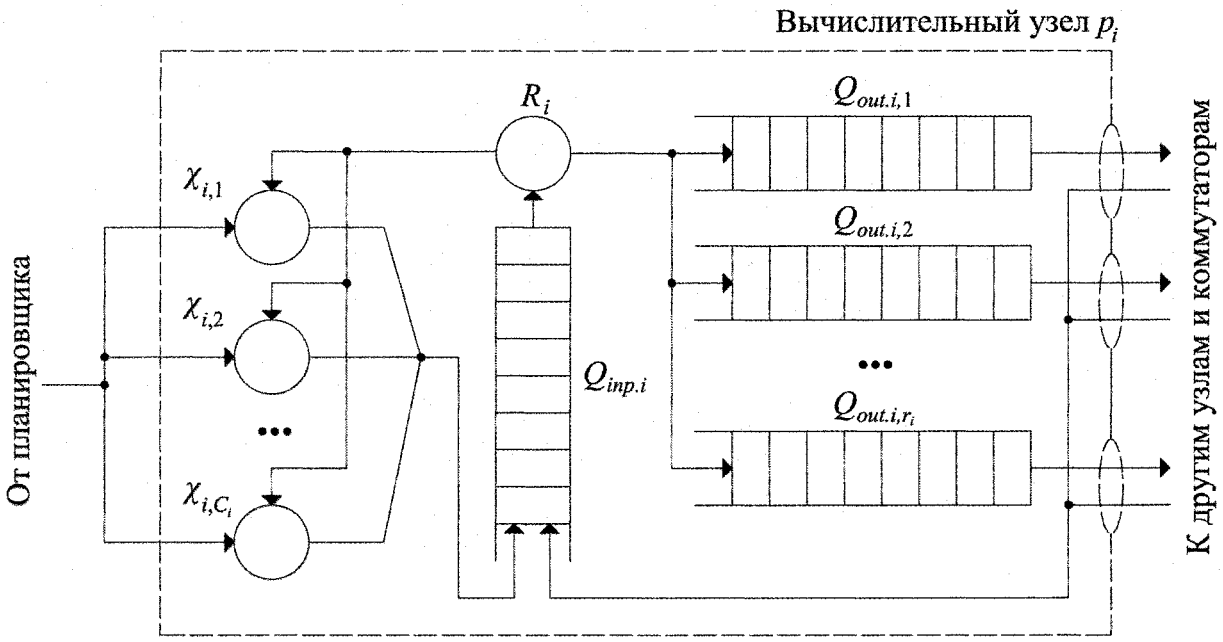

Рис. 2. Имитационная схема вычислительного узла p_i

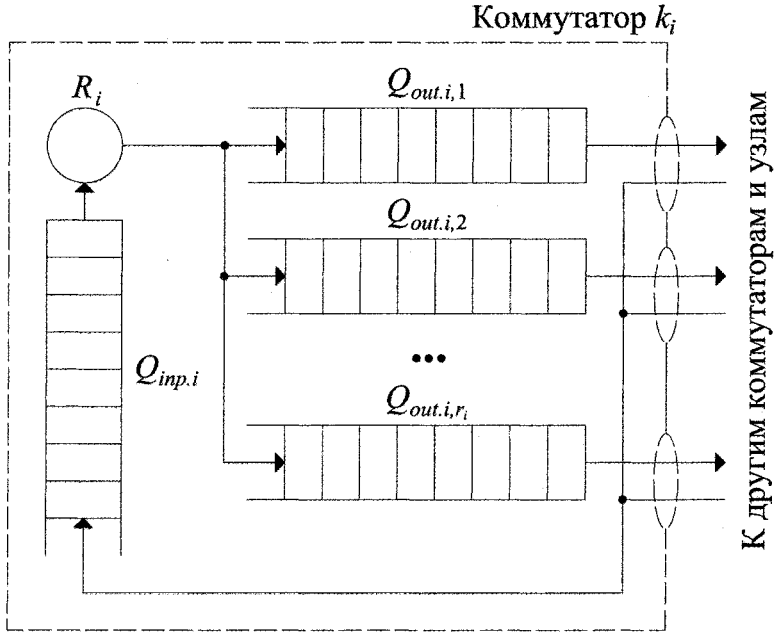

Рис. 3. Имитационная схема коммутатора k_i

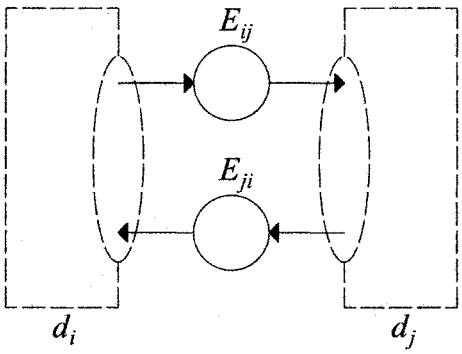

Рис. 4. Схема работы дуплексной связи L_{ij}

Обозначим в качестве $D = P \bigcup K$ множество всех сетевых устройств системы. На рис. 4 приведена схема работы дуплексной связи $L_{ij} = \{E_{ij}, E_{ji}\}$, соединяющей сетевые устройства $d_i \in D$ и $d_j \in D$. Каналы обслуживания E_{ij} и E_{ji} добавляют к времени передачи пакета задержку величиной $b(E_{ij}) = b(E_{ji})$.

Настоящая модель вычислительной системы предполагает использование принципа коммутации пакетов при передаче данных. Пусть g задает стандартный фиксированный размер одного пакета, а $h < g$ – размер его заголовочной части, тогда для передачи сообщения размера q потребуется $p = \left\lceil \frac{q}{g-h} \right\rceil$ пакетов.

Время передачи сообщения от сетевого устройства d_{i_0} к d_{i_R} вдоль пути $\alpha = (d_{i_0}, d_{i_1}, \ldots, d_{i_R})$ может быть вычислено по формуле:

$$T_{trans}(\alpha, p) = \sum_{k=0}^{R-1} \frac{g}{b(E_{d_{i_k} d_{i_{k+1}}})} + \frac{g(p-1)}{\min\limits_{k=\overline{0,R-1}} b(E_{d_{i_k} d_{i_{k+1}}})} + l_{routing} \cdot (R+1)p. \qquad (2)$$

Заметим, данная формула не учитывает возможные задержки, связанные с сетевой конкуренцией пакетов.

Настоящая работа предполагает проведение исследования алгоритмов планирования параллельных задач для случаев однородных и неоднородных вычислительных узлов, связанных гомогенной сетью. Рассматриваются следующие топологии современных кластерных вычислительных систем: двухуровневые толстые деревья, $2D$ и $3D$ торы, k-арные n-кубы, k-арные n-деревья и звезды с единственным коммутатором в центре. Все они могут быть описаны с помощью ориентированного графа (1).

2. Модель вычислительной загрузки кластера

Вычислительная загрузка кластера формируется потоком задач $J = (J_1, ..., J_m)$, помещаемых пользователями в очередь его управляющей системы. Каждая задача представляет собой параллельную неинтерактивную программу, способную работать в пакетном режиме. Ее процессы запускаются планировщиком одновременно на всех выделенных вычислительных ядрах, во время работы они обмениваются сообщениями между собой. Ресурсы, выделенные задаче, освобождаются при завершении всех ее процессов.

Каждая задача J_j характеризуется следующим набором параметров:

$$J_j = (n_j, mr_j, dr_j, \tilde{t}_j, a_j, \tau_j, CP_j), \qquad (3)$$

где n_j – количество необходимых вычислительных ядер, mr_j и dr_j – соответственно объемы оперативной и дисковой памяти в килобайтах, запрашиваемые для исполнения каждого

процесса задачи. Величина $\tilde{t}_j$ задает оценку пользователя в секундах времени выполнения программы на узлах с единичной относительной производительностью. Параметр a_j определяет время поступления задачи в очередь, а $\tau_j \in (0; \tilde{t}_j]$ – время, затрачиваемое задачей только на вычисления (без сетевых коммуникаций) при условии единичной относительной производительности всех назначенных ей вычислительных узлов. Множество CP_j задает набор коммуникационных паттернов задачи J_j.

Следующая формула позволяет вычислить t_j – оценку времени выполнения задачи с учетом производительности выделенных ей планировщиком вычислительных узлов:

$$t_j = \frac{\tilde{t}_j}{\min\limits_{i \in I_j} S_i}, \tag{4}$$

где I_j – множество номеров узлов, ядра которых были отданы задаче. Аналогичная формула применяется для корректировки значения параметра τ_j.

В рамках данной модели предполагается, что пользователь, делая оценку времени выполнения задачи, возможно, допускает ошибку, переоценивая ее по сравнению с реальным временем выполнения. Противоположный случай ошибок не рассматривается, т.к. управляющая система будет принудительно завершать задачи, превысившие лимит выделенного им времени.

Структура программы каждой параллельной задачи J_j может быть представлена следующей SPMD-моделью:

Process u**:**
for $i = 1$ to q_j do
{
    Communication(i, j);
    Computation(i, j);
}

Выполнение каждого процесса параллельной задачи представляет собой чередование фаз коммуникации и вычислений.

Коммуникационная часть каждой итерации может быть описана с помощью коммуникационного паттерна – ориентированного взвешенного графа следующего вида:

$$CP_{ij} = (\Pi_j, p_{ij}), \tag{5}$$

где $\Pi_j = \{\pi_1, \pi_2, \ldots, \pi_{n_j}\}$ – множество процессов задачи, $p_{ij} \colon \Pi_j \times \Pi_j \to \mathbb{Z}_+ \bigcup \{0\}$ – функция, определяющая вес каждой дуги, равный размеру в пакетах передаваемого сообщения. $p_{ij}(u, v)$ определяет количество пакетов сообщения, передаваемого от процесса u к процессу v на i-й итерации SPMD.

Тогда $CP_j = \{CP_{1j}, CP_{2j}, \ldots, CP_{q_j j}\}$ – множество всех коммуникационных паттернов задачи J_j.

Пусть $pred_{ij}(u) = \{v \in \Pi_j : p_{ij}(v, u) > 0\}$ – множество предшественников процесса u в орграфе (5), $succ_{ij}(u) = \{v \in \Pi_j : p_{ij}(u, v) > 0\}$ – множество его последователей. Выполнение фазы Communication(i, j) для каждого процесса u заключается в том, что сначала происходит неблокируемая рассылка сообщений всем процессам множества $succ_{ij}(u)$, а затем выполняется блокируемый прием всех сообщений от процессов $pred_{ij}(u)$.

Можно выделить два основных способа задания множества CP_j: случайная генерация и моделирование реальных программ, например, эталонных тестов, реализации быстрого

преобразования Фурье, алгоритма параллельного умножения матриц и др. Типичные коммуникационные паттерны, встречаемые в научной литературе [1, 2]: Random, Pairs, Ring, One-to-all и All-to-all.

Заметим, что в рамках настоящей модели рассматриваются только коммуникации вида «точка-точка». Моделировать коллективные операции достаточно сложно, т.к. способ их выполнения в виде совокупности точечных операций зависит от нескольких факторов: используемых алгоритмов, характеристик вычислительной системы и передаваемых сообщений. В частности, коллективные операции MPI по-разному реализованы в библиотеках разных производителей, а также в разных версиях библиотеки одного производителя. В будущем планируется расширение данной модели за счет учета коллективных операций.

Пусть функция $\gamma_j\colon \Pi_j \to P$ позволяет определить для каждого процесса задачи J_j вычислительный узел, на ядро которого он был назначен планировщиком. Для простоты положим, что каждый процесс тратит одинаковое время на выполнение вычислительной части каждой итерации SPMD-модели задачи. Тогда время выполнения вычислительной фазы Computation(i,j) процессом u можно вычислить по формуле:

$$T_{comp.ij}(u) = \frac{\tau_j}{q_j s(\gamma(u))}. \tag{6}$$

Пусть $T_{comm.ij}(u)$ – время выполнения коммуникационной фазы Communication(i,j) процессом u задачи J_j, тогда реальное время выполнения данной задачи T_j может быть вычислено по формуле:

$$T_j = \max_{u\in\Pi_j} \sum_{i=1}^{q_j} (T_{comm.ij}(u) + T_{comp.ij}(u)) = \max_{u\in\Pi_j} \{\sum_{i=1}^{q_j} T_{comm.ij}(u) + \frac{\tau_j}{s(\gamma(u))}\}. \tag{7}$$

Величина $T_{comm.ij}(u)$ зависит от сетевой конкуренции и от физического расположения процессов на вычислительных узлах, ее значение может быть вычислено в процессе симуляции работы вычислительного кластера и его управляющей системы. Заметим, что в рамках данной модели мы, в силу незначительности, пренебрегаем временем, которое тратится на разбиение сообщения на пакеты и его сборку из них.

Симуляция работы вычислительного кластера и его управляющей системы предполагает формирование потока задач. Все описанные выше количественные параметры генерируются на основе определенных законов распределений случайных величин, подобранных в результате анализа реальных трасс, собираемых на кластерных вычислительных системах (см. таблицу). Более подробную информацию можно найти в статье [3].

Таблица

Законы распределений параметров модели

Параметр	Используемый закон распределения
$\log n_j$	двухэтапное равномерное распределение с выделением классов последовательных и 2^k-задач
$\log \tau_j$	гипер-гамма распределение, параметр p которого линейно зависит от n_j
$\tilde{t}_j$	равномерное распределение на отрезке $[\tilde{\tau}_j; 2\tilde{\tau}_j]$
$a_{j+1} - a_j$	экспоненциальное распределение
mr_j, dr_j	равномерное распределение
$p_{ij}(u,v)$	экспоненциальное распределение для всех дуг, существующих в CP_{ij}

3. Алгоритм работы симулятора

В основе симулятора вычислительного кластера и его управляющей системы лежит алгоритм моделирования по принципу особых состояний.

В рамках рассматриваемой модели выделены следующие особые состояния (события):

1. Поступление новой задачи в очередь.
2. Окончание передачи пакета по сетевой связи.
3. Окончание маршрутизации пакета на сетевом устройстве.
4. Окончание вычислительной фазы процесса задачи.
5. Запуск спланированной задачи.

Для работы алгоритма моделирования по принципу особых состояний используется очередь Q_{event}, содержащая события перечисленных выше типов. Они сортируются по неубыванию времени наступления, а при равных значениях времени – в порядке приоритетности событий согласно данному списку. Это необходимо во избежание возможных конфликтов.

Обобщенная блоксхема работы алгоритма моделирования приведена на рис. 5. После

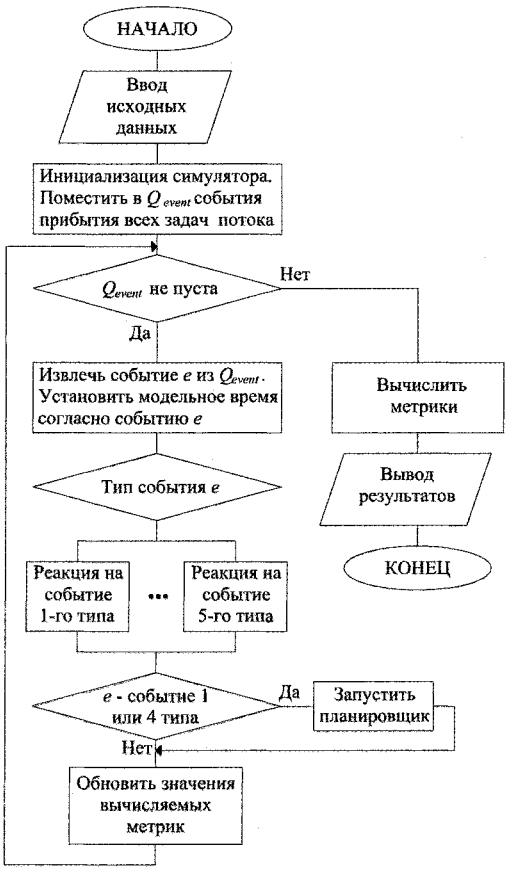

Рис. 5. Обобщенный алгоритм работы симулятора

ввода параметров модели выполняется инициализация симулятора. Затем, пока очередь

Q_{event} не пуста, происходит извлечение очередного события и его обработка, в процессе которой в Q_{event} могут добавляться новые события. Например, при обработке события окончания передачи пакета по сетевой связи может создаваться событие окончания маршрутизации этого пакета на сетевом устройстве-получателе.

При наступлении событий поступления новой задачи или окончания вычислительной фазы процесса задачи происходит запуск алгоритма планирования. Значения количественных метрик обновляются в конце каждой итерации цикла моделирования и окончательно вычисляются по его завершению.

4. Система критериев и метрик сравнения алгоритмов планирования

С целью учета всех аспектов эффективности работы алгоритмов планирования задач для кластерных вычислительных систем была построена система критериев и метрик их сравнения. Она включает следующие критерии:

1. Производительность расписаний. Содержит метрики: средняя загруженность вычислительных ядер узлов кластера U_{cores}, потеря производительности кластера CL, максимальное время завершения задачи C_{max}, среднее время ожидания задач в очереди $\overline{t}_{wait}$ и среднее ограниченное замедление задач $\overline{s}_{lim}$. Метрики U_{cores}, CL и C_{max} характеризуют непосредственно производительность расписания, поэтому они более важны для исследования, чем $\overline{t}_{wait}$ и $\overline{s}_{lim}$, которые имеют косвенный характер.

2. Используемость оперативной и дисковой памяти. Включает метрики $\overline{m}$ и $\overline{d}$, определяющие соответственно средние загруженности оперативной и дисковой памяти вычислительных узлов. Позволяет выявить процент неиспользованных ресурсов.

3. Сбалансированность загрузки узлов. Определяет метрики Du, Dm и Dd, обозначающие соответственно дисперсии загруженности ядер, оперативной и дисковой памяти вычислительных узлов. Демонстрирует честность алгоритма планирования по отношению к узлам.

4. Гарантированность обслуживания задач. Включает метрики максимального времени ожидания задач в очереди $t_{wait.max}$ и максимального ограниченного замедления задач $s_{lim.max}$. Данный критерий имеет особую важность в случае, если кластер используется в рамках системы реального времени.

5. Честность по отношению к задачам. Содержит метрику Dt_{wait} – дисперсию времени ожидания задач в очереди. Позволяет оценить степень равноправности задач с точки зрения алгоритма планирования.

6. Коммуникационный критерий. Включает следующие метрики: среднее суммарное расстояние $\overline{SD}$ между вычислительными ядрами, назначенными процессам параллельных задач, средняя дисперсность задач $\overline{JD}$ и среднее время блокировки сообщений $\overline{MB}$. Данный критерий позволяет оценить сетевую конкуренцию и степень фрагментации параллельных задач.

При исследовании различных алгоритмов планирования нас, прежде всего, интересует производительность составляемых ими расписаний. Поэтому при анализе алгоритмов, в первую очередь, будет рассматриваться первый критерий системы, остальные – имеют вторичный характер.

Анализ алгоритмов планирования по определенному критерию представляет собой сравнение для различных алгоритмов графиков зависимостей метрик этого критерия от величины системной загрузки L и выбор лучшего варианта для того или иного случая. При этом используются усредненные значения метрик на 100-1000 различных сгенерированных потоках параллельных задач.

Системная загрузка определяется как отношение суммарного объема вычислительной работы всех задач к объему работы, который кластер может потенциально выполнить (при полной загруженности) за время до появления в управляющей системе последней задачи:

$$L = \frac{\sum\limits_{j=1}^{m} n_j \tilde{\tau}_j}{\sum\limits_{i=1}^{n} S_i C_i \cdot \max\limits_{j=\overline{1,m}} \{a_j\}} \cdot 100\%. \tag{8}$$

Данная величина позволяет характеризовать вычислительную насыщенность потока работ, поступающих в управляющую систему вычислительного кластера.

Приведем формулы для вычисления основных метрик критерия производительности. Обозначим в качестве l количество моментов времени, когда в процессе моделирования наступают определенные события, $t_1, t_2, ..., t_l$ – сами моменты времени. При этом гарантируется, что никакие параметры модели и метрики не меняются на интервале времени (t_k, t_{k+1}) $\forall k = \overline{1, l-1}$. Будем также считать, что если мы вычисляем параметр модели или метрику в момент времени t_k, то это выполняется после того, как все события модели, назначенные на это время, были обработаны и выполнен цикл планирования задач. Обозначим $Q(t_k)$ – множество задач, находящихся в очереди планировщика в момент времени t_k.

Средняя загруженность вычислительных ядер узлов кластера может быть вычислена по формуле:

$$U_{cores} = \frac{\sum\limits_{i=1}^{n} \overline{u}_i}{n} \cdot 100\%, \tag{9}$$

в свою очередь средняя загруженность каждого отдельного узла определяется выражением

$$\overline{u}_i = \frac{\sum\limits_{k=1}^{l-1} u_i(t_k)(t_{k+1} - t_k)\delta_k}{\sum\limits_{k=1}^{l-1} (t_{k+1} - t_k)\delta_k}, \tag{10}$$

где

$$\delta_k = \begin{cases} 1, & |Q(t_k)| \neq 0, \\ 0, & \text{иначе.} \end{cases} \tag{11}$$

Величина потери производительности вычислительного кластера может быть вычислена следующим образом:

$$CL = \frac{\sum\limits_{k=1}^{l-1} \sum\limits_{i=1}^{n} S_i F_{ik} (t_{k+1} - t_k)\sigma_k}{\sum\limits_{k=1}^{l-1} (t_{k+1} - t_k)\sigma_k \cdot \sum\limits_{i=1}^{n} S_i C_i} \cdot 100\%, \tag{12}$$

где F_{ik} – количество свободных ядер на узле p_i в момент времени t_k, а σ_k определяется по формуле:

$$\sigma_k = \begin{cases} 1, & |Q(t_k)| \neq 0 \wedge \sum\limits_{i=1}^{n} F_{ik} < |\mathrm{X}|, \\ 0, & \text{иначе.} \end{cases} \tag{13}$$

Заметим, что формулы (9) и (12) соответственно не учитывают отсутствие загруженности ресурсов вычислительных узлов и потерю производительности в случае пустой очереди задач планировщика. Это позволяет более объективно судить о состоянии вычислительной системы.

5. Исследуемые алгоритмы планирования

К числу алгоритмов планирования, которые будут экспериментально исследованы с помощью симулятора вычислительного кластера и его управляющей системы, относятся сочетания алгоритмов выбора задачи из очереди Most Processors First Served Scan и Backfill со следующими методами назначения, учитывающими топологию вычислительной системы [1, 2, 4, 5, 6]:

1. Paging, Multiple Buddy System, Adaptive None-Contiguous Allocation, Greedy Available Busy List, Minimizing Contention, Minimizing Contention Incremental, Manhattan Median, NEP (для топологий торов);
2. Contiguous Allocation, Quasi-Contiguous Allocation (k-арные n-деревья);
3. алгоритмы сортировки узлов (толстые деревья, звезды);
4. модифицированный вариант алгоритма Minimizing Contention (для произвольных топологий).

А также со следующими методами назначения, не принимающими ее во внимание [3]: First Fit, Best Fit, Fastest Node Fist, Random First.

Заключение

В данной работе описывается симулятор вычислительного кластера, учитывающий его топологию, коммуникационные задержки при передаче данных, а также многопроцессорность вычислительных узлов. Он будет использован для исследования алгоритмов планирования задач.

Приведена имитационная схема и модель кластера, описывается модель его вычислительной загрузки, сформирована система критериев и метрик сравнения алгоритмов планирования. Также определен набор алгоритмов планирования, которые будут исследованы с помощью данного симулятора.

Статья рекомендована к публикации программным комитетом международной научной конференции «Параллельные вычислительные технологии 2010».

Работа проводилась при финансовой поддержке Федерального агентства по образованию в рамках реализации ФЦП «Научные и научно-педагогические кадры инновационной России» на 2009-2013 гг. (государственный контракт № П2039).

Литература

1. Moore, S.Q. The Effects of Network Contention on Processor Allocation Strategies / S.Q. Moore, M.N. Lionel // Proceedings of the 10th International Parallel Processing Symposium. – Washington, DC: EEE Computer Society, 1996. – P. 268 – 273.
2. Bani-Mohammad, S. An efficient processor allocation strategy that maintains a high degree of contiguity among processors in 2D mesh connected multicomputers / S. Bani-Mohammad,

M. Ould-Khaoua, I. Abaneh // Proceedings of ACS/IEEE International Conference on Computer Systems and Applications, AICCSA. – 2007. – P. 934 – 941.

3. Полежаев, П.Н. Исследование алгоритмов планирования параллельных задач для кластерных вычислительных систем с помощью симулятора / Полежаев, П.Н. // Параллельные вычислительные технологии (ПАВТ'2010): тр. междунар. конф. – Челябинск, 2010. – С. 287 – 298.
4. Bender, M.A. Communication-Aware Processor Allocation for Supercomputers / M.A. Bender, D.P. Bunde, E.D. Demaine // Lecture Notes in Computer Science. – 2005. – V. 3608/2005. – P. 169 – 181.
5. Cheng, C. Improving Performance of Mesh-connected Multicomputers by Reducing Fragmentation / C. Cheng, P. Mohapatra // J. of Parallel and Distributed Computing. – 1998. – V. 52(1). – P. 40 – 68.
6. Pascual, J.A. Effects of Topology-Aware Allocation Policies on Scheduling Performance / J.A. Pascual, J. Navaridas, J. Miguel-Alonso // Lecture Notes in Computer Science. – 2009. – V. 5798/2009. – P. 138 – 156.

Полежаев Петр Николаевич, аспирант, кафедра математического обеспечения информационных систем, Оренбургский государственный университет, peter.polezhaev@mail.ru.

Поступила в редакцию 20 мая 2010 г.