УДК 004.382.2

Алгоритмы отказоустойчивого управления ресурсами пространственно-распределённых вычислительных систем[1]

А.Ю. Поляков, О.В. Молдованова, А.А. Пазников,
М.Г. Курносов, С.Н. Мамойленко, А.В. Ефимов

В статье рассматриваются вопросы организации отказоустойчивого функционирования пространственно-распределённых вычислительных систем (ПРВС) в мультизадачных режимах. Предложена модель функционирования ПРВС, состоящих из множества подсистем. Разработаны алгоритмы функционирования распределённой очереди задач ПРВС. Для территориально-сосредоточенных вычислительных подсистем созданы алгоритмы управления ресурсами при обслуживании пользовательских задач, представленных параллельными масштабируемыми программами. Развит программный инструментарий отказоустойчивого выполнения программ на подсистемах ПРВС. Приведены результаты моделирования алгоритмов на мультикластерной ПРВС (GRID-модель).

Ключевые слова: распределённые вычислительные системы, параллельное мультипрограммирование, MPI, GRID, мультикластерные системы, диспетчеризация, отказоустойчивость.

1. Введение

В настоящее время при решении сложных задач науки и техники значительную роль играют пространственно-распределённые вычислительные системы (ПРВС). К ним относятся мультикластерные вычислительные и GRID-системы, которые представляют собой макроколлективы рассредоточенных средств обработки информации (подсистем), взаимодействующих через локальные и глобальные сети связи [1].

В архитектурном плане подсистема ПРВС представляет собой *кластерную вычислительную систему (ВС)* – композицию множества элементарных машин (ЭМ) и сети связей между ними. При создании подсистем могут быть использованы как стандартные промышленные, так и специально созданные устройства. *Элементарная машина* является основным структурным и функциональным компонентом кластерной ВС и формируется из локального коммутатора (ЛК), процессора и памяти. Кластерная ВС функционирует под управлением системного программного обеспечения [1], включающего, в том числе, систему управления ресурсами (СУР), например, TORQUE [2], Altair PBS Pro, SLURM [3], HTCondor и др.

Основным назначением ПРВС является обработка (решение) задач. *Задача* – это требование выполнить параллельную программу на ресурсах ПРВС. Описание необходимых для решения задачи ресурсов ПРВС формируется пользователем и задаётся *ресурсным запросом*, включающем: параллельную программу (созданную, например, в модели передачи сообще-

[1] Работа выполнена при поддержке Совета по грантам Президента Российской Федерации (проект МД-2620-2014.9) и Российского фонда фундаментальных исследований (гранты № 12-07-00188, 12-07-0106, 12-07-00019).

ний) и паспорт задачи (написанный, например, на языке Job Submission Description Language – JSDL).

Процесс решения задач на ПРВС организован в пакетном режиме и предусматривает следующие шаги:

1) размещение задачи пользователя в очереди задач. В ПРВС функционирует распределённая очередь задач, объединяющая локальные очереди задач СУР подсистем ПРВС. При этом задачи могут поступать как в глобальную очередь (один поток задач), так и в локальные очереди СУР подсистем ПРВС (множество потоков задач);

2) выделение необходимых ресурсов ПРВС и определение времени начала выполнения параллельной программы;

3) выполнение параллельной программы на выделенных ресурсах подсистемы ПРВС. При этом предусмотрена возможность периодического сохранения состояния вычислительного процесса в контрольных точках (КТ);

4) сохранение результатов выполнения параллельной программы.

В случае аварийного завершения программы из-за отказов вычислительных ресурсов, задача повторно помещается в очередь СУР и, после выделения необходимых ресурсов, выполнение параллельной программы возобновляется с использованием наиболее актуальной КТ. Если актуальной КТ не существует, то выполнение программы начинается заново.

В процессе решения задач мониторинг за состоянием ресурсов ПРВС и вычислительного процесса реализуется специализированными программными средствами, в основу которых положены алгоритмы самодиагностики.

В связи с большемасштабностью, территориальной удалённостью подсистем и динамическим характером состава и загрузки пространственно-распределённых ВС, актуальной является проблема эффективного и отказоустойчивого функционирования их ресурсов. В рамках указанной проблемы в данной работе рассмотрены нижеследующие задачи:

1. разработка алгоритмического и программного инструментария, позволяющего осуществлять (суб)оптимальный выбор подсистемы ВС (шаг 1 процедуры обработки задач);

2. организация обработки масштабируемых задач на подсистемах таких ВС, позволяющая эффективно загрузить ресурсы подсистем ВС (шаг 2 процедуры обработки задач);

3. развитие существующих средств отказоустойчивого выполнения параллельных программ (шаг 3, процедуры обработки задач).

2. Модель функционирования пространственно-распределённых вычислительных систем

Рассмотрим модель пространственно-распределённой ВС, состоящей из H подсистем $S = \{1, 2, 3, \ldots, H\}$. Каждая подсистема $i \in S$ характеризуется количеством n_i элементарных машин $E_i = \{1, 2, 3, \ldots, n_i\}$. На подсистеме функционирует СУР, которая в любой момент времени t характеризуется множеством $B_i(t) \subseteq E_i$ занятых ЭМ и количеством $Q_i(t)$ задач в её локальной очереди.

Подсистемы ВС взаимодействуют между собой через каналы передачи данных. Граф логических связей между подсистемами обозначим через $G = (S, U)$, где $U \subseteq S \times S$ – логические связи между подсистемами. Дуга $(i, j) \in U$ в графе означает, что подсистема i может передавать задачи для решения на подсистеме j. Вершины j, смежные вершине i, образуют её локальную окрестность $L(i) = \{j \in S \mid (i, j) \in U\}$ (рис. 1). В состав программного обеспечения ПРВС входит инструментарий мониторинга, позволяющий, в частности, оценить производительности каналов передачи данных между подсистемами. Обозначим через функцию $\theta(i, j, s)$ ($[\theta(i, j, s)] = \text{с}$) время передачи сообщения размером s байт между подсистемами i и j.

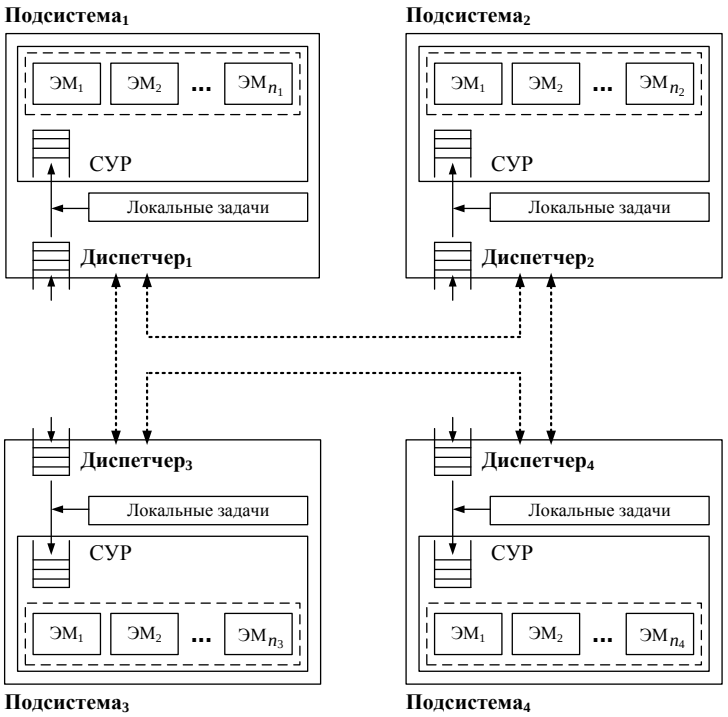


Рис. 1. Пример локальных окрестностей диспетчеров:
$H = 4$, $L(1) = \{2, 3\}$, $L(2) = \{1, 4\}$, $L(3) = \{1, 4\}$, $L(4) = \{2, 3\}$

На вход системы поступает поток $F = (1, 2, 3, \ldots, m, \ldots)$ задач. Каждая задача описывается ресурсным запросом m, который характеризуется временем a_m его поступления в систему и представляет собой кортеж $\langle c_m, p_m, d_m \rangle$, где:

1) $c_m = \{ \langle r_k, t_k, w_k \rangle, k \in \{1, 2, \ldots, |c_m|\} \}$ – множество допустимых конфигураций ресурсов подсистемы, каждая из которых представлена композицией $\langle r, t, w \rangle$ ранга r параллельной программы (количеством ЭМ, необходимым для её выполнения), времени t её выполнения и приоритета w данной конфигурации по отношению к другим конфигурациям, указанным в ресурсном запросе;

2) p_m – штраф за задержку решения задачи в единицу времени, определяющий приоритет данной задачи;

3) $d_m = \{ \langle s_k, h_k \rangle, k \in \{1, 2, \ldots, |d_m|\} \}$ – описание входных данных параллельной программы, состоящее из пар вида $\langle s_k, h_k \rangle$, где s_k – размер k-го входного файла ($[s_k]$ = байт), $h_k \in S$ – номер подсистемы, на которой расположен этот файл.

Определим также функции:

1) $\text{rank}(m)$ множества допустимых рангов задачи $m \in F$ – $\text{rank}(m) = \{ r \mid \exists\, t, w : \langle r, t, w \rangle \in c_m \}$;

2) $\text{time}(m, r)$ времени решения задачи m с рангом r – $\text{time}(m, r) = t : r \in \text{rank}(m)$ и $\exists\, w : \langle r, t, w \rangle \in c_m$;

3) $\text{weight}(m, r)$ веса конфигурации $r \times t$ ресурсов ПРВС задачи m: $\text{weight}(m, r) = w\ \exists\, t : \langle r, t, w \rangle \in c_m$.

Для каждой задачи $m \in F$ требуется выбрать конфигурацию $\langle r_m, t_m \rangle$, $r_m \in \text{rank}(m)$, $t_m = \text{time}(m, r)$ ресурсов ПРВС и время τ_m начала её обработки. Также требуется определить подсистему $j = x_m \in S$ и подмножество $e_m \subseteq E_j$, $|e_m| \geq r_m$ её ЭМ, которые будут выделены для решения задачи на временно́м отрезке $[\tau_m, \tau_m + t_m]$. В качестве ограничений часто выступает требование монопольного доступа к выделенным ресурсам подсистемы:

$$\forall\, m, n \in F\ x_m = x_n \Rightarrow |[\tau_m, \tau_m + t_m] \cap [\tau_n, \tau_n + t_n]| \cdot |e_m \cap e_n| = 0.$$

В общем случае проблема планирования является трудноразрешимой. Поэтому в настоящее время ведутся активные исследования по созданию приближённых методов её решения.

3. Алгоритмы децентрализованной диспетчеризации задач в пространственно-распределённых вычислительных системах

3.1. Обзор предшествующих работ

Существующие средства поиска подсистем для решения задач в ПРВС являются централизованными. Основной недостаток такого подхода состоит в том, что отказ централизованного диспетчера задач приводит к отказу всей ПРВС. Применение централизованных диспетчеров задач вносит существенные ограничения на масштабируемость ПРВС.

На сегодняшний день из числа программных пакетов диспетчеризации задач в ПРВС следует выделить: GridWay [4], AppLeS [5], GrADS [6], Nimrod/G [7], Condor-G [8], WMS [9]. Пакет GridWay широко распространён и достаточно активно развивается. Цель диспетчеризации в GridWay – минимизация времени обслуживания задач. Учитываются зависимости по данным между задачами, а также реализована возможность их миграции между подсистемами ПРВС. AppLeS реализует диспетчеризацию на уровне программ, что существенно снижает универсальность диспетчера. Пакет GrADS, как и GridWay, поддерживает миграцию задач. В Nimrod/G реализована экономическая модель, обеспечивающая баланс между условными поставщиками и потребителями ресурсов. В Condor-G максимизируется пропускная способность системы и реализовано представление задач в виде ориентированных ациклических графов.

В настоящее время активно развиваются мультиагентные системы, такие как AgentScape [10], DIRAC [11]. На подсистемах функционируют агенты, которые запрашивают задачи из глобальной очереди. Наличие глобальной очереди является основным их недостатком. Предлагаемые мультиагентные методы диспетчеризации могут быть использованы и в ПРВС, однако в них в недостаточной степени учитывается динамический характер состава и загрузки подсистем и каналов связи.

Важное место занимают работы, связанные с разработкой инструментария диспетчеризации задач, представленных в виде ориентированных графов. Существующие системы данного класса, такие как Pegasus [12], Taverna [13], Triana [14], ASKALON [15], ICENI [16], Kepler [17] и др., являются централизованными. Помимо этого, большинство диспетчеров направлено на применение в узкоспециализированных областях.

В работе предлагается использовать децентрализованный подход к диспетчеризации задач в ПРВС. На каждой подсистеме ВС находится диспетчер, взаимодействующий с ограниченным количеством других диспетчеров (локальная окрестность). Коллектив диспетчеров совместно выбирает ресурсы для запуска задач. Такой подход характеризуется низкой трудоёмкостью, что позволяет использовать его в большемасштабных ПРВС.

Разработаны алгоритмы децентрализованной диспетчеризации, учитывающие переменный характер загрузки ресурсов ПРВС [18].

3.2. Алгоритмы децентрализованной диспетчеризации

В работе предложены децентрализованные алгоритмы диспетчеризации, обеспечивающие отказоустойчивое распределение задач по подсистемам ПРВС.

Рассмотрим пространственно-распределённую ВС. На каждой её подсистеме помимо стандартных СУР функционирует дополнительный компонент стека системного программного обеспечения, называемый «диспетчером». Коллектив диспетчеров связан согласно графу G.

Пользователь направляет задачу $m \in F$ одному из известных ему диспетчеров $i \in S$ ПРВС. Диспетчер фиксирует время a_m поступления задачи и производит в своей локальной окрестности $L(i) \cup \{i\}$ поиск (суб)оптимальной подсистемы j^*, на которой будет выполняться её решение. После завершения обработки реализуется доставка соответствующих выходных файлов на подсистему i.

Алгоритм диспетчера (*DecentSched*):

1. Обращение диспетчера i к системе мониторинга ресурсов ПРВС с целью получения текущих значений параметров n_j, $B_j(t)$, $Q_j(t)$, $j \in L(i) \cup \{i\}$.

2. Формирование множества $A(i, m)$ допустимых подсистем

$$A(i,m) = \{ j \mid n_j \geq \max_{r \in \mathrm{rank}(m)} r,\ j \in L(i) \cup \{i\} \},$$

имеющих количество ЭМ не ниже требуемого для выполнения параллельной программы.

3. Из окрестности $A(i, m)$ диспетчера i выбирается подсистема j^* с минимальным значением целевой функции $C(j)$, $j \in A(i, m)$:

$$j^* = \arg\min_{j \in A(i,m)} C(j),\ C(j) = \begin{cases} \dfrac{\theta_j}{\theta_{\max}} + \dfrac{\mu_{\max}}{\mu_j} + \dfrac{\omega_j}{\omega_{\max}}, & \text{если } \mu_j < \max\limits_{r \in rank(m)} r \text{ или } \mathrm{Q}_j(t) > 0, \\ \dfrac{\theta_j}{\theta_{\max}}, & \text{иначе,} \end{cases}$$

где $\theta_j = \sum_{\langle s,h \rangle \in d_m} \theta(h, j, s)$ – время доставки файлов задачи до подсистемы j; $\theta_{\max} = \max\limits_{j \in A(i,m)} \{\theta_j\}$; $\mu_j = n_j - B_j(t)$, $\mu_{\max} = \max\limits_{j \in A(i,m)} \{\mu_j\}$, $\omega_j = Q_j(t) / n_j$ – количество задач в очереди, приходящееся на одну ЭМ подсистемы j; $\omega_{\max} = \max\limits_{j \in A(i,m)} \{\omega_j\}$.

4. На выбранную подсистему j осуществляется доставка файлов набора d_m, а задача направляется в локальную очередь соответствующей СУР.

5. Диспетчер i с интервалом времени Δ повторяет шаги 1 – 4 поиска подсистемы j' для задачи в очереди диспетчера j^*.

6. Если для найденной подсистемы выполняется условие $C(j^*) - C(j') > \upsilon$, то выполняется миграция задачи в очередь подсистемы j' (задача удаляется из очереди j^*).

Экспериментальное сравнение (рис. 2) созданного алгоритма децентрализованной диспетчеризации и централизованного алгоритма, используемого в пакете GridWay, показывает, что алгоритм *DecentSched* обладает аналогичной эффективностью. С другой стороны, *DecentSched* устойчив к отказам подсистем ВС и позволяет обеспечить живучее функционирование большемасштабных пространственно-распределённых вычислительных систем.

4. Алгоритмы обработки наборов масштабируемых (moldable) задач на подсистемах пространственно-распределённых ВС

Децентрализованный диспетчер, распределяющий задачи по подсистемам ПРВС, формирует в каждой из них очереди $f_i(t) \subseteq F$ из $Q_i(t)$ задач. Также в эти очереди пользователи локальных СУР могут помещать задачи, минуя диспетчер распределённой очереди задач. Работа каждой СУР рассматривается независимо как проблема организации функционирования кластерной ВС в режиме обработки набора $f = f_i(t)$ задач. Поэтому в рамках данного раздела индекс i подсистемы ПРВС будем опускать. Каждая СУР включает: очередь, планировщик и среду решения пользовательских задач.

Наиболее распространёнными алгоритмами, применяемыми при планирования решения задач, являются [19]: First-Come-First-Served (FCFS), Shortest-Job-First (SJF), Longest-Job-First (LJF). Также получили распространение технологии Backfilling и Gang Scheduling [20]. Первая ориентирована на повышение эффективности обработки задач с малым временем решения, а вторая – для снижения общего времени отклика системы. Существуют алгоритмы, учитывающие несколько критериев при планировании решения задач [21].

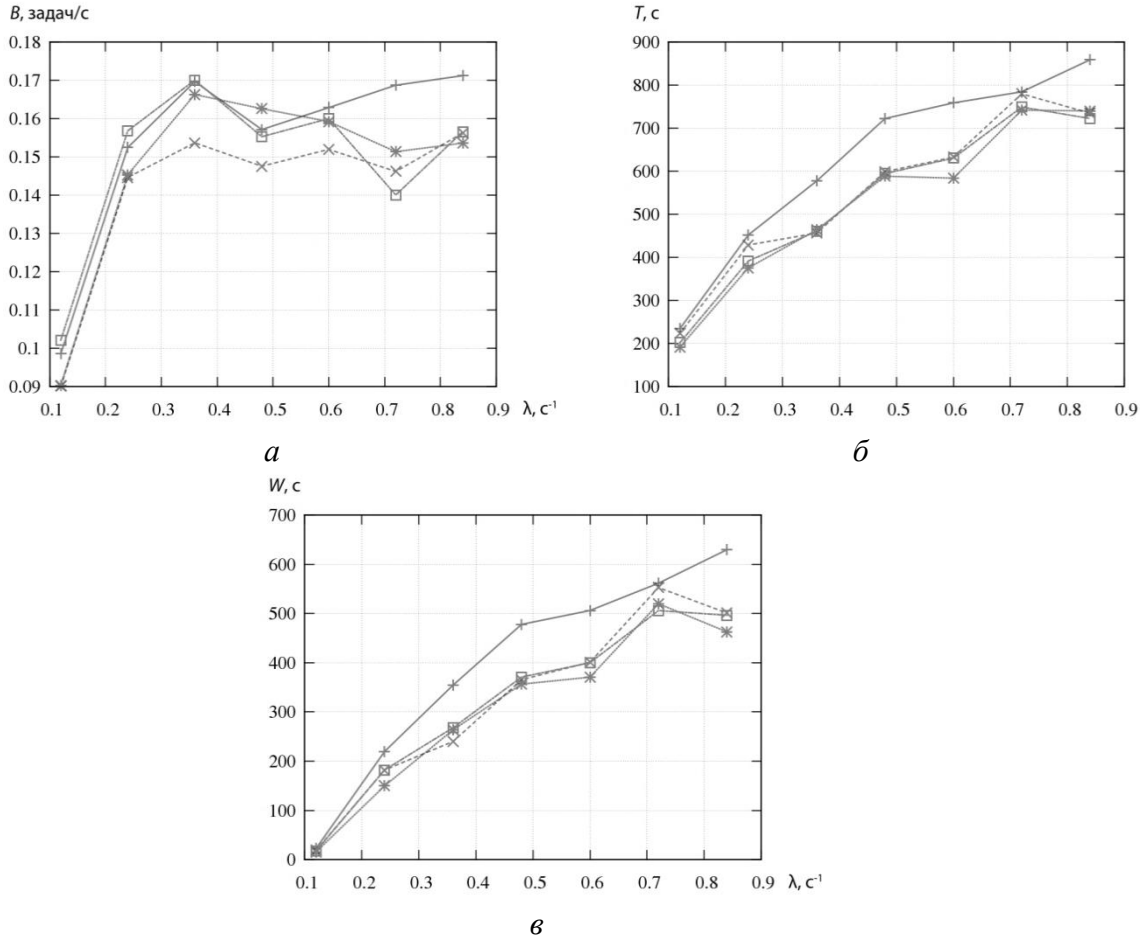


Рис. 2. Сравнение алгоритма *DecentSched* децентрализованной диспетчеризации с централизованным алгоритром диспетчера GridWay
а – пропускная способность *B* системы, *б* – среднее время *T* обслуживания задачи,
в – среднее время *W* пребывания задачи в очереди
– *DecentSched*, обслуживание потока задач на одной подсистеме,
– *DecentSched*, полный граф, – *DecentSched*, 2*D*-тор,
– GridWay

Для организации эффективной обработки задач на ресурсах ВС предложены стохастические алгоритмы планирования, обеспечивающие многокритериальную оптимизацию расписаний. В качестве критериев оптимизации рассматриваются время решения всех задач набора f и суммарный штраф за задержку их решения.

Процедура распределения ресурсов разбита на чередующиеся этапы полного и частичного планирования. На каждом из них для задач набора f формируется (суб)оптимальное расписание, определяющее подмножество $\delta_l \subseteq f$, обработка которого может быть произведена в i-ом этапе планирования. При этом между l и $l+1$ этапами полного планирования допускается произвольное количество этапов частичного планирования. Далее рассмотрим каждый из них подробнее.

1. На некотором этапе i полного планирования, производящегося в момент времени t, все ресурсы системы должны быть свободны: $B(t) = \varnothing$. На время решения задач не накладывается ограничений. После завершения определяется время T_l следующей процедуры полного планирования

$$T_l^* = t + \max_{m \in \delta_i} \text{time}(m, r_m),$$

где r_m – ранг параллельной программы, выбранный из множества допустимых конфигураций задачи m.

2. *Частичное планирование*, инициированное между l и l +1 этапами полного планирования в момент времени $t < T_l^*$, предполагает, что доступно лишь ограниченное подмножество $E \setminus B(t)$ ЭМ подсистемы, освободившихся в результате досрочного завершения задач из δ_i. Все ресурсы, выделенные на данном шаге, должны быть освобождены до начала следующего полного планирования в момент времени T_i^*.

Рассмотрим обобщённую процедуру планирования, выполняющуюся в момент времени t. Задано ограничение T^* на время решения задачи: для полного планирования $T^* = \infty$, для частичного – $T^* = T_l^* - t$. Для планирования доступно $E^* = E \setminus B(t)$ ЭМ. Ресурсные запросы задач удовлетворяют физическим ограничениям подсистемы i: $\forall m \in f \, \forall r \in \text{rank}(m)$ $0 < r \le n$, $\text{time}(m, r) > 0$, $\text{weight}(m, r, t) > 0$.

Для каждой задачи $m \in f^* = \{m' \in f \mid \exists\, r \in \text{rank}(m): r \le |E^*|, \text{time}(m, r) \le T^*\}$ необходимо выбрать: время τ_m начала её решения и количество $r_m \in \text{rank}(m) : r_m < |E^*|$ параллельных ветвей, при этом время решения задачи при выбранном ранге r_m не должно превышать заданного ограничения T^*. Также требуется определить подмножество $e_m \subseteq E^*$, $|e_m| \ge r_m$ ЭМ подсистемы, на котором будет осуществляться решение задачи. Набор кортежей $\Theta(t) = \{\langle \tau_m, r_m, e_m \rangle \mid m \in f^* \}$ представляет собой *расписание решения задач*. Его характеристиками являются время $T(\Theta(t))$ реализации и суммарный штраф $P(\Theta(t))$ за задержку решения:

$$T(\Theta(t)) = \max_{m \in f^*} \tau_m + \text{time}(m, r_m), \; P(\Theta(t)) = \sum_{m \in f^*} \left((\tau_m - a_m) \cdot p_m \right).$$

Требуется найти такое расписание Θ^* решения задач на ВС, что:

$$T(\Theta^*) = \min_{\Theta \in \Xi} T(\Theta), \; P(\Theta^*) = \min_{\Theta \in \Xi} P(\Theta) \tag{1}$$

при ограничениях:

$$\forall m \in f^*, |e_m| \ge r_m \tag{2}$$

$$\forall t' \ge t \bigcap_{m \in \Gamma(t)} e_m = \varnothing, \; \sum_{m \in \Gamma(t)} r_m \le E^*, \tag{3}$$

$$\gamma_m = \frac{w_m}{\max\limits_{r \in \text{rank}(m)} \text{weight}(m, r)}, \; \frac{1}{|f^*|} \sum_{m \in f^*} \gamma_m \ge \varepsilon, \tag{4}$$

где

Ξ – множество допустимых расписаний при обработке задач из f^*, находящихся в очереди СУР;

$\Gamma(t) = \{ m \mid m \in f^*, \tau_m \le t \le \tau_m + t_m \}$ – множество задач, решаемых в момент времени t;

γ_m – относительный вес выбранной конфигурации ресурсов задачи m;

ε – минимально допустимый суммарный относительный вес конфигураций ресурсов расписания.

Задача (1) – (4) относится к многокритериальной оптимизации.

4.1. Алгоритмы формирования укрупнённых задач

На первом этапе планирования задачи $m \in f^*$ разбиваются на подмножества таким образом, чтобы выполнялись ограничения (2) – (4) и достигался минимум функций (1). Сформировать разбиение возможно, например, с использованием алгоритмов ортогональной упаковки прямоугольников без поворотов и пересечений. При этом задача кодируется прямоугольным объектом с высотой, равной рангу задачи, и шириной, соответствующей запрашиваемому времени. В данной работе для упаковки использован алгоритм FFDH (First Fit Decrease Height). Очевидно, что в случае разбиения масштабируемых задач процедура упаковки усложняется выбором одного из допустимых размеров прямоугольника.

Формально постановка задачи первого этапа приобретает следующий вид. Решением является множество пар $W_i = \{ \{\langle r_m, b(m)\rangle \mid m \in f^* \}$, где $b(m)$ – номер укрупнённой задачи, которой принадлежит задача m. Также определена обратная функция $b^{-1}(k) \subseteq f^*$, возвращающая множество задач, входящих в укрупнённую задачу k. Требуется построить решение W^* такое, чтобы

$$T(W^*) = \min_{W \in \Xi'} T(W),\ T(W) = \sum_{j=1}^{|W|} \max_{m \in b^{-1}(j)} (\tau_m + t_m) \tag{5}$$

при ограничениях:

$$\bigcup_{k=1}^{|W|} b^{-1}(k) = f^*,\ \bigcap_{k=1}^{|W|} b^{-1}(k) = \varnothing, \forall W \in \Xi', \tag{6}$$

$$\sum_{m \in b^{-1}(k)} r_m \le \left|E^*\right|, \forall W \in \Xi,\ \forall k \in \{1, 2, ..., |W|\}, \tag{7}$$

где Ξ' – область допустимых решений задачи первого этапа.

При выборе допустимой конфигурации ресурсов подсистемы, выделяемой для задачи m, необходимо контролировать выполнение ограничения (4). Разобьём задачи из f^* на три класса: $I^0, I^1, I^2 \subseteq f^*$: $I^0 \cap I^1 \cap I^2 = \varnothing$, $I^0 \cup I^1 \cup I^2 = f^*$, где $I^0 = \{ m \mid m \in f^*, |c_m| = 1 \}$, $I^1 = \{ m \mid m \in f^*, |c_m| > 1,\ \forall r, r' \in \text{rank}(m)\ \text{weight}(m, r) = \text{weight}(m, r') \}$; $I^3 = f^* \setminus (I^0 \cup I^1)$. Тогда ограничение (4) можно представить в следующем виде:

$$\frac{1}{\left|f^*\right|}\left(\sum_{m \in I^0} \gamma_m + \sum_{m \in I^1} \gamma_m + \sum_{m \in I^2} \gamma_m\right) \ge \varepsilon \tag{8}$$

Для задач из I^0 и I^1 слагаемые выражения (8) равны соответственно $|I^0|$ и $|I^1|$. На выполнение ограничения (4) влияет лишь выбор значений параметров для задач из подмножества I^2:

Утверждение 1. *Если* $|I^0| + |I^1| \ge \varepsilon \cdot |f^*|$ *или*

$$\frac{\min\limits_{m \in I^2} \min\limits_{r \in \text{rank}(m)} \text{weight}(m, r)}{\max\limits_{m \in I^2} \max\limits_{r \in \text{rank}(m)} \text{weight}(m, r)} \ge \frac{\varepsilon \cdot \left|f^*\right| - \left|I^0\right| - \left|I^1\right|}{\left|I^2\right|},$$

то ограничение (4) *выполняется при любом выборе значений параметров для задач подмножества* I^2.

Если условия утверждения 1 не выполняется, то используем следующую процедуру. Считаем, что для задач уже выбраны значения параметров (либо удовлетворяющие ограничению (4), либо с наивысшим приоритетом).

1. Для задач подмножеств I^1 конфигурации выбираются произвольно.

2. Определяется среднее значение степени учёта приоритетов конфигураций.

3. Произвольно выбирается задача из подмножества $m \in I^2$ и новая допустимая конфигурация $\langle r, t, w\rangle \in c_m$.

4. Если изменение приоритета значений параметров задачи не приводит к нарушению условия (5), то фиксируется $r_m = r$, $t_m = t$, $w_m = w$ и осуществляется переход к шагу 2; в противном случае выбранная конфигурация задачи m не меняется.

5. Процедура продолжается до тех пор, пока не будут перебраны все задачи подмножества I^2.

4.2. Алгоритм формирования расписаний по набору укрупнённых задач

Алгоритм на основе метода цепей Монте-Карло − итерационный. На каждом шаге случайным образом выбираются значения $\langle r, t, w\rangle \in c_m$ так, чтобы удовлетворять ограничению (4), и формируются укрупнённые задачи алгоритмом FFDH. Шаги повторяются до тех пор, пока на заданном количестве итераций не будет найдено ни одного разбиения с лучшим значением функции (5).

После того как сформированы укрупнённые задачи W^*, необходимо определить одну из них $k^* \in W^*$, которой ресурсы ВС будут выделены в первую очередь. Для каждой укрупнённой задачи k вычисляется время решения T_k и штраф P_k за задержку её решения:

$$\forall k \in \{1,2,...,\left|f^*\right|\}, T_k = \max_{m \in b^{-1}(k)}(t_m), \mathrm{P}_k = \sum_{m \in b^{-1}(k)} p_m(t - a_m),$$

где t – момент времени, в который осуществляется процедура планирования; a_m – время поступления задачи на ПРВС.

Для решения на ПРВС выбирается укрупнённая задача k^*, удовлетворяющая следующему условию

$$k^* = \arg\min_{k \in \{1,2,....,|W^*|\}} \frac{T_k}{P_k},$$

справедливость которого доказана в [22].

Тогда $\delta = b^{-1}(k^*)$ и задачи $m \in \delta$ удаляются из очереди: $f(t+\Delta t) = f(t) \setminus \delta$. ЭМ подсистемы распределяются по принципу First-In First-Out (FIFO). Представим множества δ и E^* в виде векторов: $\pi = \langle \pi_1, \pi_2, \ldots, \pi_l\rangle$, $\rho = \langle \rho_1, \rho_2, \ldots, \rho_k\rangle$, где $l = |\delta|$, $k = |E^*|$. Тогда для задачи π_m, $m \in \{1, 2, \ldots l\}$ выделяются ЭМ с номерами

$$e_m = \{\rho_\alpha, \rho_{\alpha+1}, ..., \rho_{\alpha+r_m}\}, \alpha = \sum_{j=1}^{m-1} r_m .$$

Если выполняется процедура полного планирования, то вычисляется время следующего полного планирования

$$T_i^* = t + \max_{m \in \delta} \tau_m .$$

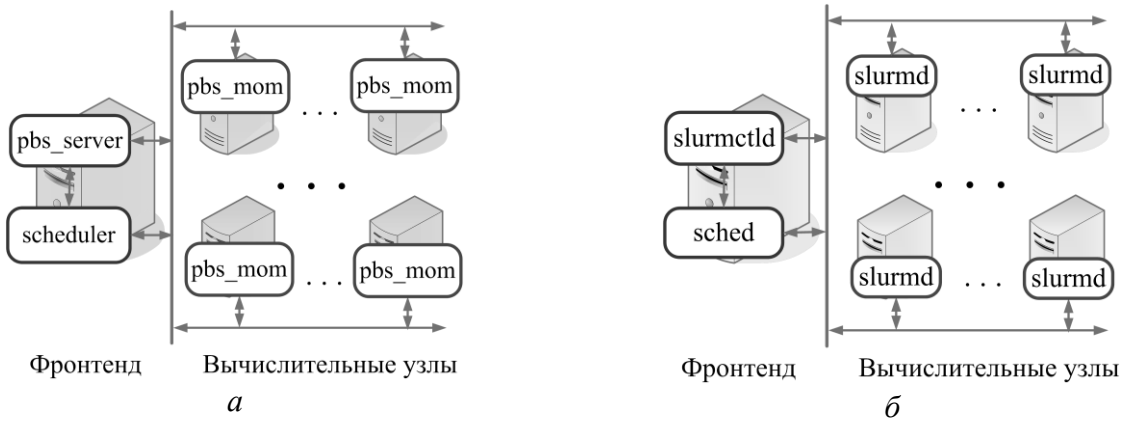
Задачи $m \in \delta$ распределяются на соответствующие элементарные машины e_m. Следующая процедура полного планирования выполняется в момент времени T_i^*, следующая процедура частичного планирования производится при освобождении более χn_i узлов подсистемы.

5. Анализ и оптимизация систем управления ресурсами

5.1. Программное обеспечение управления ресурсами кластерных ВС

Как было сказано ранее, для управления ресурсами подсистем ПРВС используются СУР. В рамках данной работы проведён анализ наиболее распространённых свободных реализаций данного класса программного обеспечения: TORQUE – Terascale Open-Source Resource and QUEue Manager и SLURM – Simple Linux Utility for Resource Management. В рамках данного раздела под ВС будем понимать подсистему ПРВС.

Функциональные схемы СУР TORQUE и SLURM схожи друг с другом (рис. 3, *а*, *б*).



а *б*

Рис. 3. Функциональные схемы СУР: *а* – TORQUE, *б* – SLURM

Каждый вычислительный узел ВС оснащается локальным компонентом СУР: PBS MOM – Machine Oriented Miniserver (TORQUE) или SLURM Daemon (SLURM), который осуществляет управление ресурсами данного узла. Также предусмотрен централизованный компонент: PBS Server (TORQUE) или SLURM Control Daemon (SLURM), который поддерживает очередь задач, взаимодействует с планировщиком ресурсов, а также при помощи соответствующего локального компонента координирует работу узлов ВС и осуществляет мониторинг их состояния. Для обеспечения отказоустойчивости в SLURM предусмотрено наличие дополнительного экземпляра центрального компонента, дублирующего состояние главного. В качестве планировщика ресурсов в обеих СУР предусмотрена возможность использования как встроенного компонента, так и внешнего модуля, например, Maui Cluster Scheduler или Moab Workload Manager.

5.2. Решение масштабируемых задач

Известно, что свойством масштабируемости обладают до 98 % задач, решаемых на высокопроизводительных ВС [23]. Анализ существующих менеджеров и систем планирования загрузки ресурсов распределённых ВС показывает, что большинство из них имеют лишь базовую поддержку масштабируемых (moldable) задач, которая не позволяет определить предпочтения (или приоритеты) на выделение конкретных рангов вычислительных подсистем.

Коллективом авторов была предложена модификация Maui, реализующая базовые функции по обслуживанию масштабируемых задач. Для активации режима обработки масштабируемых задач в конфигурационном файле данного планировщика предусмотрена опция EnableMoldableJobsSheduling.

Суть работы планировщика Maui заключается в формировании расписания решения задач из очереди локальной СУР и резервирование ресурсов для задач, у которых время начала решения меньше, чем интервал планирования. Последнее возможно, поскольку все задачи обладают временем старта и прерывания, что позволяет своевременно доставлять исходные данные задачи на ЭМ выделенной подсистемы.

Непосредственно цикл планирования начинается в случае возникновения следующих ситуаций:

- изменилось состояние задачи или ЭМ;
- достигнута граница резервирования;
- поступает соответствующая команда;
- с момента окончания предыдущего цикла прошло время, определённое как максимальное.

Задачи рассматриваются в порядке поступления в очередь или согласно fairshare политик.

В процессе цикла планирования для каждой задачи из очереди либо находится подсистема доступных свободных ресурсов, либо необходимое количество ресурсов резервируются в ближайшее доступное по расписанию время. В первом случае планировщик отправляет СУР команду «начать решение соответствующей задачи». Во втором случае задача продолжает ожидание в очереди, пока наступит время резервирования или необходимые ресурсы будут получены ранее. Последнее возможно при досрочном завершении решения предшествующих задач или в результате выполнения алгоритма backfilling.

Множество c_m допустимых конфигураций ресурсов для задач определяются в ресурсном запросе с помощью специального флага TRL в следующем формате:

$$\#\text{PBS } -\text{l trl}=r_1@t_1@w_1:r_2@t_2@w_2: \ \ldots\ :r_k@t_k@w_k,$$

где r_i – запрашиваемое количество узлов (или ppn, если в системе всего один узел), t_i – оценка времени решения задачи, w_i – вес варианта конфигурации, отражающий пожелания пользователя. В процессе планирования в качестве базовой конфигурации выбирается вариант, обладающий максимальным весом на единичный ресурс – $w_i / (r_i * t_i)$. При выполнении алгоритма backfilling рассматриваются все варианты конфигураций ресурсов в порядке уменьшения веса.

Для тестирования предложенной версии Maui был проведён ряд экспериментов. Поток задач в них генерировался для тестовой системы с 8 процессорными ядрами на основе модели рабочей загрузки, предложенной в работе [23]. Результаты показывают, что учёт свойства масштабируемости задач позволил на 15 % сократить среднее время ожидания задач в очереди.

5.3. Организация среды исполнения параллельных программ

При запуске задачи m ей выделяется подмножество элементарных машин e_m, принадлежащих вычислительным узлам ВС. Среди этих узлов выделяется один, называемый «головным». На головном узле выполняется последовательность команд, указанная в паспорте задачи. Одной из целей этих операций является создание инфраструктуры для организации параллельных вычислений. Современные реализации моделей распределённого параллельного программирования предусматривают наличие среды времени выполнения (runtime environment – RTE), которая решает следующие задачи:

1) запуск (*launch*) процессов, выполняющих код параллельных ветвей;
2) связывание (*connect*) ветвей друг с другом;
3) перенаправление ввода-вывода (*io forwarding*) ветвей на головной узел;
4) управление (*control*) ветвями программы.

Функции RTE могут осуществляться непосредственно системой параллельного программирования или СУР через интерфейс управления процессами Process Management Interface (PMI).

Для развёртки RTE большинство современных СУР предоставляют механизмы массового удалённого запуска процессов на узлах ВС – Remote Execution Service (RES).

Запуск заданного процесса на удалённом узле осуществляется в два этапа: 1) установка соединения с этим узлом; 2) непосредственно запуск указанной программы. Так как второй шаг осуществляется полностью локально удалённым узлом, то с точки зрения *масштабирования* ВС наиболее ресурсоёмким является первый шаг.

Существует два основных подхода к организации RES: *централизованный*, при котором запуск всех процессов производится с головного узла, и *распределённый*, при котором другие узлы подмножества e_m берут на себя часть функций по организации RES. Во втором случае структура сетевых соединений имеет вид дерева.

Анализ RES современных СУР показывает, что большинство из них, в частности TORQUE, используют централизованный подход к организации RES. В TORQUE доступ к сервису доступен через прикладной интерфейс программирования (API), который реализуется библиотекой *libtorque* или через интерфейс командной строки (программа pbsdsh).

Псевдокод массового запуска процессов на выделенном наборе e_m ЭМ с использованием TORQUE API приведён ниже:

```
call tm_init()
foreach n in e_m do
   call tm_spawn(n, command)
done
call tm_finalize()
```

SLURM – единственная из свободно-распространяемых реализаций СУР, обеспечивающая распределённый массовый запуск процессов на узлах ВС. Доступ к данному сервису осуществляется через интерфейс командной строки (программа *srun*).

Анализ древовидной структуры соединений, формируемых компонентами SLURM, показал, что в ней присутствует дисбаланс (рис. 4, *а*). Авторами предложен оптимизированный вариант алгоритма RES для данной СУР, обеспечивающий более сбалансированную загрузку (рис. 4, *б*) вычислительных узлов набора e_m. Псевдокод предложенного алгоритма приведён ниже, на его вход поступает количество n вычислительных узлов ВС, на которых данный узел обеспечивает развёртку RTE, и ширина w дерева, описывающего структуру сетевых соединений. Результатом является вектор *span*, распределяющий нагрузку между данным узлом и его w потомками. Алгоритм реализован в пакете SLURM и включён в его официальный дистрибутив, начиная с версии 14.11.

```
function set_span(n, w)
   span = (0, 0, …. 0)
   if ( n <= w)
      return span
   l = n
   while l > 0 do
      for i = 0 to w do
         if span[i] = 0 then
            l = l – 1
         fi
         if (w – i  ≥ l) then
            if span[i]  ≠ 0 then
               span[i] = span[i] + l
            fi
            l = 0
         elif l ≤ w then
            span[i] = span[i] + l
            l = 0
         else
            span[i] = span[i] + w
            l = l – w
         fi
      done
   done
   return span;
```

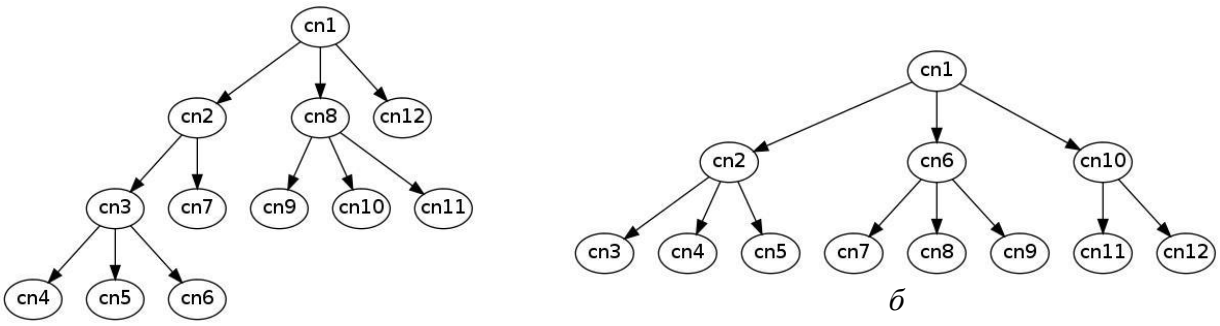


Рис. 4. Структура соединений, устанавливаемых компонентами SLURM, реализующими RES:
а – оригинальный алгоритм; *б* – модифицированный алгоритм.
cn1 – cn12 – доменные имена вычислительных узлов,
составляющих выделенную подсистему ЭМ.

6. Программный инструментарий отказоустойчивого выполнения параллельных программ

6.1. Средства отказоустойчивого выполнения параллельных программ

Несмотря на высокую надёжность компонентов большемасштабных ПРВС, среднее время между частичными отказами в них составляет несколько дней. Это определяет необходимость применения средств обеспечения отказоустойчивого выполнения параллельных программ [24], основанных на формировании контрольных точек (КТ). Одним из наиболее развивающихся программных продуктов в данной области является пакет DMTCP [25, 26].

Данный программный пакет реализован на уровне системных библиотек и использует синхронный подход к формированию контрольных точек для распределённых (в частности, параллельных) программ. DMTCP не требует изменения исходного кода защищаемой программы, её перекомпиляции или пересборки. Также не требуется модификации ядра операционной системы (ОС). Для сбора информации о выполняющейся программе используются интерфейсы ОС, в частности, специализированная файловая система /proc. В тех случаях, когда данные о некотором фрагменте состояния нельзя получить стандартными средствами, производится сохранение последовательности действий, необходимых для его восстановления. Для этой цели применяется техника, предусматривающая перехват системных вызовов операционной системы узла ВС.

На каждую вычислительную группу (набор процессов, связанных с некоторой параллельной программой) создаётся отдельный процесс-координатор, который обеспечивает синхронность формирования распределённой КТ. DMTCP автоматически обнаруживает попытку запуска процессов на удалённых узлах с применением протокола SSH. Благодаря этому сохраняется контроль над всеми ветвями параллельной программы, если программа запускается *без применения RES* СУР.

При формировании распределённой КТ каждый процесс сохраняет своё состояние в отдельном файле. В частности, записывается содержимое памяти программы, состояние таблицы файлов, информация об активных сетевых соединениях и транзитных сетевых данных. Транзитными называются данные, отправленные удалённой стороной, но ещё не доставленные получателю в момент начала формирования КТ.

После того как все процессы сохранили своё состояние, процесс-координатор формирует shell-скрипт, содержащий последовательность действий, необходимых для запуска вычислений из данной КТ на *исходном* наборе вычислительных узлов.

6.2. Интеграция средств отказоустойчивого выполнения программ и систем управления ресурсами ВС

В настоящее время актуальной является разработка алгоритмов и программных компонентов, позволяющих интегрировать пакет DMTCP с СУР, в первую очередь, TORQUE и SLURM. Непосредственное взаимодействие с исполняемой параллельной программой осуществляет локальный компонент СУР. Поэтому он представляет наибольший интерес при решении рассматриваемой задачи.

На рис. 5 показана процедура запуска параллельной программы, соответствующей некоторой задаче *m*, системой TORQUE. Управляющий компонент СУР через соответствующий локальный компонент головного узла размещает информацию о наборе ЭМ e_m, выделенных для решения задачи *m*. Система TORQUE передаёт эти данные системе параллельного программирования через файл, имя которого определяется в среде окружения. Система SLURM размещает информацию о e_m непосредственно в среде окружения. Также подготавливаются файлы, предназначенные для стандартного вывода программы. После этого на головном узле инициируется выполнение команд, указанных в паспорте задачи и предназначенных для развёртки RTE и запуска параллельной программы. Обычно RTE предусматривает наличие одного служебного процесса на каждом из узлов, который обслуживает все прикладные процессы, реализующие ветви параллельной программы.

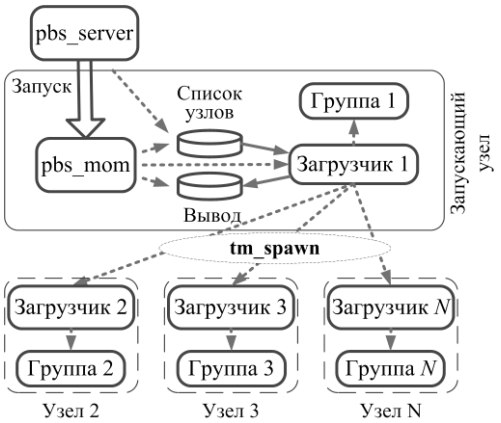


Рис. 5. Функциональная схема TORQUE

Анализ функциональных структур рассмотренных СУР позволяет сформулировать нижеследующие подзадачи, решение которых требуется для интеграции с ними пакета DMTCP.

1. Поддержка пакетом DMTCP RES, предоставляемых СУР TORQUE и SLURM.

2. Распознавание и специальная обработка пакетом DMTCP служебных файлов СУР. Особая обработка служебных файлов связана с тем, что их имена обычно изменяются при каждом запуске программы под контролем СУР.

3. Обнаружение пакетом DMTCP ресурсов, выделенных СУР, при возобновлении параллельных программ из контрольных точек. Как показано ранее, в текущей конфигурации DMTCP обеспечивает возобновление программы только на исходном наборе вычислительных узлов.

Для решения первой задачи выполнено расширение существующего механизма перехвата удалённых вызовов. Для поддержки RES, реализованной в СУР TORQUE, разработана «обёртка» (wrapper) вокруг функции tm_spawn, её псевдокод приведён ниже:

```
function tm_spawn(<arglist>)
  libpath = call find_libtorque()
  spawn_orig = call dyn_load(libpath,"tm_spawn")
  <arglist_new> = call MODIFY(<arglist>)
  return spawn_orig(<arglist_new>)
```

Функция *find_libtorque* выполняет обнаружение библиотеки *libtorque*, используя данные ядра ОС GNU/Linux (файл /proc/self/maps). Функция *dyn_load* выполняет загрузку оригинальной функции *tm_spawn*, которая была «перекрыта» разработанной обёрткой. Указатель на оригинальную функцию из библиотеки *libtorque* сохраняется в переменной *spawn_orig*. Функция *MODIFY* модифицирует аргументы функции *tm_spawn* так, чтобы удалённый процесс был также запущен под управлением DMTCP. Наконец, выполняется вызов оригинальной функции *tm_spawn*.

Для поддержки RES, реализованного в системе SLURM, аналогичным образом были модифицированы обёртки системных вызовов ОС семейства *exec*.

Из двух рассматриваемых СУР только TORQUE требует дополнительной обработки служебных файлов, так как они располагаются в специализированной директории (обычно, /var/spool/torque) и не могут быть воссозданы на этапе возобновления программы. Для обработки служебных файлов TORQUE была модифицирована подсистема DMTCP, обеспечивающая сохранение состояния таблицы файлов. Файлы, принадлежащие СУР, определяются на основании информации о её конфигурации через известные интерфейсы. Восстановление служебного файла TORQUE, хранящего список узлов выделенной подсистемы ВС, выполняется следующим образом. Соединение с файлом, созданным PBS MOM и содержащим узлы новой вычислительной подсистемы, закрывается. Вместо него создаётся подключение к исходному node-файлу, сохранённому в КТ.

Содержимое выходных файлов копируется в новые файлы, созданные PBS MOM. Это обеспечивает прозрачность (незаметность) формирования КТ для пользователя: после успешного завершения программы ему будет предоставлен финальный вариант выходных данных.

Для решения задачи обнаружения ресурсов, выделенных СУР, разработана программа *dmtcp_discover_rm* (DMTCP Discover Resource Manager), которая выполняет: 1) обнаружение ресурсов, выделенных СУР; 2) их сопоставление с ресурсами, требуемыми для возобновляемой задачи по алгоритму, псевдокод которого приведён в листинге ниже.

```
function dmtcp_map(C)
    cmap = call load_map(C)
    nmap = call query_rm()
    n = cmap.nodes[cmap.hnp]
    n' = nmap.nodes[nmap.hnp]
    if n'.proc_num < n.proc_num then
        call fail_to_map()
        return error
    fi
    call assign(n', n)
    n'.proc_num = n'.proc_num – n.proc_num
    call sort(cmap.nodes)
    n'' = nil
    foreach n in cmap.nodes do
        call sort(nmap.nodes)
        f = false
        foreach n' in nmap.nodes do
            if n'.proc_num ≥ n.proc_num then
                f = true
                n'' = n'
                break
            else
                break
            fi
```

```
        done
        if flag then
            call assign(n", n)
            n".proc_num = n".proc_num – n.proc_num
        else
            call fail_to_map
            return error
        fi
    done
    return cmap
```

На вход функции *dmtcp_map* поступает КТ *C*, из которой производится восстановление. Функция *load_map* загружает из КТ *C* информацию об отображении процессов параллельной программы по исходным узлам. Функция *query_rm* автоматически определяет СУР, под управлением которой производится восстановление и запрашивает у неё набор узлов, выделенный для возобновляемой программы. Обе функции записывают в поле *hnp* индекс узла, являющегося головным. Алгоритм учитывает связи между процессами параллельной программы, в частности, процессы, исходно выполнявшиеся на одном узле, отображаются на один узел новой подсистемы. Данное требование обусловлено тем, что такие процессы могли использовать для взаимодействия общую память. Запуск удалённых процессов параллельной программы на новой подсистеме узлов осуществляется с использованием RES, соответствующей СУР.

Вызов программы *dmtcp_discover_rm* осуществляется из shell-скрипта, формируемого процессом-координатором в момент формирования КТ. Ссылка на этот скрипт размещается в паспорте новой задачи, которая ставится в очередь СУР. После запуска этой задачи на новом подмножестве ЭМ будет выполнена реконструкция состояния исходной параллельной программы до соответствующего моменту создания КТ. После этого выполнение продолжится с учетом всех промежуточных результатов, полученных программой к моменту формирования соответствующей КТ.

Разработанные алгоритмы и программные средства были реализованы в пакете DMTCP и включены в его официальный дистрибутив, начиная с версии 2.1.

7. Заключение

В работе предложены новые алгоритмы организации отказоустойчивого функционирования пространственно-распределённых вычислительных систем.

Предложены децентрализованные алгоритмы диспетчеризации. Экспериментально подтверждено, что они обладают аналогичной эффективностью по сравнению с централизованными аналогами. Преимуществом предложенных алгоритмов является отказоустойчивость и масштабируемость относительно числа подсистем в ПРВС.

Разработаны стохастические алгоритмы планирования ресурсов подсистем ПРВС при решении масштабируемых задач, обеспечивающие многокритериальную оптимизацию расписаний. В качестве критериев оптимизации рассматриваются время решения всех задач набора и штраф за задержку решения, который может быть определён как совокупность административного, пользовательского и системного приоритетов.

Развиты компоненты стека системного программного обеспечения ПРВС. В планировщике Maui создан базовый функционал, позволяющий реализовать обработку масштабируемых задач.

Проведена оптимизация алгоритма взаимодействия между компонентами системы SLURM управления ресурсами подсистем ПРВС. Модифицированный алгоритм включён в официальную версию SLURM.

Инструментарий DMTCP организации отказоустойчивого выполнения параллельных программ развит средствами интеграции со свободными менеджерами ресурсов TORQUE и SLURM. Предложенные изменения также включены в официальный дистрибутив.

Литература

1. *Хорошевский В.Г.* Распределённые вычислительные системы с программируемой структурой// Вестник СибГУТИ. 2010. №2 (10). С. 3−41.
2. Torque Resource Manager [Электронный ресурс]. Режим доступа: http://www.adaptivecomputing.com/products/open-source/torque/ (дата обращения 03.09.2014).
3. SLURM: Simple Linux Utility for Resource Management, A. Yoo, M. Jette, and M. Grondona, Job Scheduling Strategies for Parallel Processing, volume 2862 of Lecture Notes in Computer Science, pages 44-60, Springer-Verlag, 2003.
4. *Huedo E., Montero R.S., Llorente I.M.* A framework for adaptive execution on grids // Software – Practice and Experience (SPE). 2004. Vol. 34. P. 631–651.
5. *Berman F., Wolski R., Casanova H.* Adaptive computing on the grid using AppLeS // IEEE Trans. on Parallel and Distributed Systems. 2003. Vol. 34. P. 369–382.
6. *Cooper K., Dasgupta A., Kennedy C.K.* [et al]. New Grid Scheduling and Rescheduling Methods in the GrADS Project // Proc. of the 18th International Parallel and Distributed Processing Symposium (IPDPS'04). 2004. Vol. 34. P. 199–206.
7. *Buyya R., Abramson D., Giddy J.* Nimrod/G: An architecture for a resource management and scheduling system in a global computational Grid // Proc. of the 4th International Conference on High Performance Computing in Asia-Pacific Region. 2000. P. 283–289.
8. *Frey J., Tannenbaum T., Livny M.* [et al.] Condor-G: A computation management agent for multi-institutional grids // Cluster Computing. 2001. Vol. 5. P. 237–246.
9. *Andreetto P., Borgia S., Dorigo A.* Practical approaches to grid workload and resource management in the EGEE project / // In CHEP '04: Proceedings of the Conference on Computing in High Energy and Nuclear Physics. 2004. Vol. 2. P. 899–902.
10. *Wijngaards N., Overeinder B., Steen M., Brazier F.* Supporting internet-scale multi-agent systems // Data Knowledge Engineering. 2002. Vol. 41. P. 229–245.
11. *Caron E., Garonne V., Tsaregorodtsev A.* Evaluation of Meta-scheduler Architectures and Task Assignment Policies for High Throughput Computing // Technical report № 5576. Institut National de Recherche en Informatique et en Automatique. 2005. 16p.
12. *Deelman E., Singh G., Su M.-H.* [et al.] Pegasus: A framework for mapping complex scientific workflows onto distributed systems // Scientific Programming. 2005. Vol. 13(3). P. 219–237.
13. *Hull D., Wolstencroft K., Stevens R.* [et al.] Taverna: a tool for building and running workflows of services / // Nucleic Acids Research. 2006. Vol. 34. P. 729–732.
14. *Matthew I., Shields M., Wang I., Philp R.* Grid Enabling Applications Using Triana // In Workshop on Grid Applications and Programming Tools. 2003. 11p.
15. *Fahringer T., Prodan R., Duan R.* [et al.] ASKALON: A Grid Application Development and Computing Environment // 6th IEEE/ACM International Workshop on Grid Computing. 2005. P. 122–131.
16. *Young L., Mcgough S., Newhouse S., Darlington J.* Scheduling Architecture and Algorithms within the ICENI Grid Middleware // In UK e-Science All Hands Meeting. 2003. P. 5–12.
17. *Altintas I., Berkley C., Jaeger E.* [et al.] Kepler: An Extensible System for Design and Execution of Scientific Work-flows // International Conference on Scientific and Statistical Database Management. 2004. P. 21–23.
18. *Kurnosov M., Paznikov A.* Efficiency analysis of decentralized grid scheduling with job migration and replication // ACM International Conference on Ubiquitous Information Management and Communication. 2013. 7 p.

19. *Feitelson D.G.,* [*et. al*]. Theory and practice in parallel job scheduling // Job Scheduling Strategies for Parallel Processing. 1997. Vol. 1291. P. 1 – 34.
20. *Shmueli E.,Feitelson D.G.* Backfilling with lookahead to optimize the packing of parallel jobs. J. Parallel & Distributed Comput. 2005. Vol. 65. Iss. 9. P. 1090 – 1107.
21. *Cirne W., Grande C., Berman F.* When the herd is smart aggregate behavior in the selection of job request. IEEE Transactions in Parallel and Distributed Systems. 2003. Vol. 14. P. 181 – 192.
22. *Мамойленко С.Н., Ефимов А.В.* Алгоритмы планирования решения масштабируемых задач на распределённых вычислительных системах. Вестник ГОУ ВПО «СибГУТИ». 2010. № 2. С. 66 – 78.
23. *Cirne W., Berman F.* A model for moldable supercomputer jobs. 15th Intl. Parallel & Distributed Processing Symp. 2001 URL: `http://cseweb.ucsd.edu/~walfredo/papers/moldability-model.pdf` (дата обращения: 29.09.2014).
24. *Elnozahy, E.N., [et. al.]* A survey of rollback-recovery protocols in message-passing systems. ACM Computing Surveys. 2002. Vol. 34. N. 3. P. 375 – 408.
25. *Ansel J., Arya K., Cooperman G.* DMTCP: Transparent Checkpointing for Cluster Computations and the Desktop. IEEE International Parallel and Distributed Processing Symposium (IPDPS'09). 2009. 12 p.
26. *Поляков А.Ю.* О восстановлении программ из контрольной точки / А.Ю. Поляков. Вестник ЮУрГУ. Серия «Математическое моделирование и программирование». 2010. № 35(211). С. 91 – 103.

Статья поступила в редакцию 01.09.2014;

Поляков Артём Юрьевич
к.т.н., доцент кафедры вычислительных систем СибГУТИ, (630102, Новосибирск, ул. Кирова, 86), тел. (383) 269-82-75, e-mail: `artpol@csc.sibsutis.ru`, м.н.с. Лаборатории вычислительных систем ИФП СО РАН (630090 Новосибирск, пр. Ак. Лаврентьева, 13) тел. (383) 330-56-26, e-mail: `artpol@isp.nsc.ru`.

Молдованова Ольга Владимировна
к.т.н., доцент, доцент кафедры вычислительных систем СибГУТИ, (630102, Новосибирск, ул. Кирова, 86) тел. (383) 269-82-75, e-mail: `ovm@csc.sibsutis.ru`

Пазников Алексей Александрович
к.т.н., доцент кафедры вычислительных систем СибГУТИ, (630102, Новосибирск, ул. Кирова, 86) тел. (383) 269-82-93, e-mail: `apaznikov@gmail.com`.

Курносов Михаил Георгиевич
к.т.н., доцент кафедры вычислительных систем СибГУТИ, (630102, Новосибирск, ул. Кирова, 86) тел. (383) 269-82-75, e-mail: `mkurnosov@gmail.com`.

Мамойленко Сергей Николаевич
д.т.н., доцент, профессор кафедры вычислительных систем СибГУТИ, (630102, Новосибирск, ул. Кирова, 86) тел. (383) 269-83-82, e-mail: `msn@sibsutis.ru`.

Ефимов Александр Владимирович
к.т.н., доцент кафедры вычислительных систем СибГУТИ, (630102, Новосибирск, ул. Кирова, 86) тел. (383) 269-82-93, e-mail: `efimov@cpct.sibsutis.ru`.

Algorithms of fault-tolerant resources management of geographically distributed computer systems

A. Y. Polyakov, O.V. Moldovanova, A.A. Paznikov, M.G. Kernosov, S.N. Mamoilenko, A.V. Efimov

This paper considers the issues of fault-tolerant operation of geographically distributed computer systems (GDCS) in multitask modes. The model of GDCS operation consisting of many subsystems was proposed. Algorithms of fault-tolerant distributed job queue operation were developed. Algorithms of resource management for the GDCS subsystems were created. The program tools of fault-tolerant execution of parallel programs on GDCS subsystems were developed and implemented. Simulation results of the algorithms on the multicluster GDCS (GRID model) were presented.

Keywords: distributed computer systems, parallel multiprogramming, MPI, GRID, multicluster systems, dispatching, fault-tolerance, and self-diagnosis.