УДК 004.451.24

А.Ю. Поляков, О.В. Молдованова

ДЕЛЬТА-ОПТИМИЗАЦИЯ КОНТРОЛЬНЫХ ТОЧЕК ВОССТАНОВЛЕНИЯ ПАРАЛЛЕЛЬНЫХ ПРОГРАММ[1]

Рассмотрены подходы к дельта-оптимизации контрольных точек восстановления параллельных программ, реализуемых на вычислительных системах. Описаны и исследованы алгоритмы дельта-сжатия контрольных точек, основанные на хешировании. Предложен адаптивный подход к дельта-оптимизации, сочетающий преимущества различных видов дельта-сжатия, и показана его эффективность для набора прикладных программ.

Ключевые слова: *отказоустойчивость*, *контрольные точки восстановления программ*, *распределённые вычислительные системы.*

Распределенные вычислительные системы (ВС) являются основным инструментом высокопроизводительной обработки информации [1]. Современные ВС [2] являются большемасштабными, они формируются из десятков и сотен тысяч вычислительных ядер и обладают производительностью от TeraFLOPS до PetaFLOPS.

Согласно статистическим данным [3 – 5], среднее время безотказной работы (λ^{-1}) вычислительных узлов (ВУ) распределенных ВС лежит в промежутке 10^4 – 10^6 ч (λ – интенсивность потока отказов одного ВУ). Но даже при использовании таких высоконадежных компонентов время между частичными отказами (отказами отдельных ВУ) в большемасштабных ВС в среднем составляет несколько дней. Это ставит под вопрос осуществимость решения трудоемких задач, представленных параллельными программами с количеством ветвей, близким к числу узлов ВС.

Для обеспечения отказоустойчивости распределенных ВС используют аппаратурную или (и) программную избыточность. Аппаратурная избыточность предполагает одновременное использование альтернативных элементов в структуре распределенной ВС для решения одной задачи (см. например, технологии RAID, Channel Bonding и т. п.). Программная избыточность предполагает формирование служебной информации, которая может быть применена для устранения последствий частичных отказов ресурсов ВС. Наиболее распространенной реализацией данного подхода является периодическое сохранение промежуточных состояний параллельных программ (ПП) в контрольных точках (КТ) восстановления. Такие КТ используются для перезапуска ПП в случае отказов ресурсов ВС.

Для формирования распределенной КТ параллельной программы каждый процесс, входящий в ее состав, сохраняет свое состояние в локальной КТ. Целостной распределенной КТ [6] называется набор локальных КТ, позволяющих сформировать допустимое состояние ПП. Существует два основных подхода к созданию распределенных КТ: синхронный (координированный) и асинхронный. При син-

[1] Работа выполнена в рамках интеграционного проекта № 113 СО РАН, при поддержке РФФИ (гранты № 09-07-13534, 09-07-90403, 10-07-00157, 08-07-00022), Совета по грантам Президента РФ для поддержки ведущих научных школ (грант НШ-5176.2010.9) и в рамках государственного контракта № 02.740.11.0006 с Минобрнауки РФ.

хронном подходе все процессы ПП сохраняют свое состояние одновременно, что гарантирует целостность КТ, но приводит к повышенной нагрузке на систему ввода-вывода распределенной ВС. При асинхронном подходе каждый процесс создает КТ независимо от других. Поэтому при восстановлении необходимо выполнять поиск целостного состояния программы на основе набора локальных КТ, созданных в различные моменты времени. При этом возможно возникновение «эффекта домино», когда происходит запуск ПП из начального состояния.

Анализ существующих средств отказоустойчивого выполнения параллельных программ показывает, что для создания распределенных КТ, как правило, используется синхронный подход [7 – 9]. В связи с вышесказанным актуальной является задача снижения накладных расходов, вносимых средствами отказоустойчивости ВС, за счет уменьшения объема локальных КТ. Для решения данной проблемы может быть использована технология дельта-сжатия [10 – 12], обеспечивающая исключение фрагментов состояния параллельной программы, сохраненных в предшествующих КТ. Применение дельта-сжатия для уменьшения объема и времени создания КТ будем называть дельта-оптимизацией.

В работе проведен анализ существующих алгоритмов дельта-сжатия, а также предложен новый адаптивный подход к дельта-оптимизации, сочетающий преимущества известных видов дельта-сжатия.

1. Применение дельта-сжатия для оптимизации контрольных точек восстановления параллельных программ

Методы и алгоритмы, реализуемые в параллельных программах, характеризуются различным использованием доступной памяти. Например, многие итерационные методы лишь частично модифицируют обрабатываемые данные в процессе работы. Поэтому КТ часто содержат дублирующуюся информацию, исключение которой может значительно снизить объемы передаваемых и хранимых данных.

Различают два вида дельта-сжатия: инкрементное и дифференциальное. Первое предусматривает сохранение фрагментов текущей КТ, измененных относительно предыдущей КТ (рис. 1, *а*), второе – сохранение фрагментов текущей КТ, модифицированных относительно некоторой базовой КТ (рис. 1, *б*).

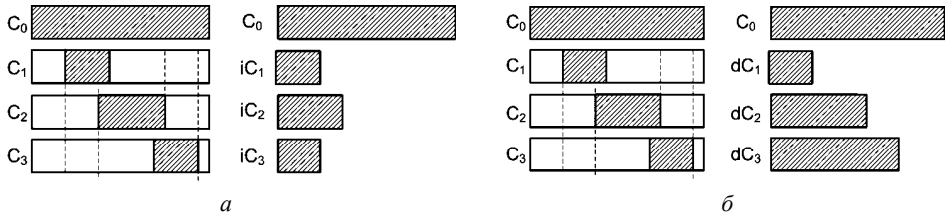

Рис. 1. Контрольные точки: *а* – инкрементные; *б* – дифференциальные; – фрагменты, которые сохраняются в подсистеме ввода-вывода; – фрагменты, которые не сохраняются в подсистеме ввода-вывода

Преимуществом дифференциального дельта-сжатия является то, что для формирования результирующей КТ требуется только базовая КТ и одна дифференциальная КТ (ДКТ). Это делает возможным сокращение объемов хранимых данных за счет удаления устаревших дифференциальных КТ.

Инкрементные КТ (ИКТ), по сравнению с дифференциальными, формируются относительно более актуального состояния и в большинстве случаев будут иметь меньший объем. С другой стороны, данный подход характеризуется большим (по сравнению с ДКТ) временем формирования результирующей КТ и необходимостью хранить все созданные ИКТ.

Существуют два основных подхода к реализации дельта-сжатия КТ: страничный [12] и основанный на хешировании [10, 11]. Первый имеет ряд ограничений, связанных с размером страницы памяти и аппаратурными возможностями процессора. Второй подход является более универсальным, он использует хеш-функции с малой вероятностью коллизии для обнаружения измененных фрагментов КТ. В данной работе рассмотрены алгоритмы, относящиеся к подходам, основанным на хешировании.

Пусть K_l – КТ, созданная в момент времени $l = \{1, 2, ..., L\}$, а K_{li}, $i = 1, \ldots, S(K_l)$ – i-й байт K_l, где $S(X)$ – размер структуры данных X в байтах. Разобьем K_l на R блоков, размер каждого из которых определяется применяемым подходом:

$$B_l = \{B_{lj} \mid B_{lj} = (K_{lz}, K_{l(z+1)}, \ldots, K_{l(z+m)}),\ z = \sum_{i=1}^{j-1} d_i\ ,\ m = d_j\}, \quad (1)$$

где $d = \{d_1, d_2, \ldots, d_R\}$ – размеры соответствующих блоков. Для каждого блока j вычисляется хеш-код g_{lj}, совокупность g_l хеш-кодов всех блоков КТ K_l будем называть ее сигнатурой:

$$g_l = \{g_{lj} \mid g_{lj} = h(B_{lj})\}. \quad (2)$$

В выражении (2) h – хеш-функция, ставящая в соответствие набору байт произвольной длины набор фиксированной длины, равной β байт. При этом $\beta << d_i$ $\forall i = 1, \ldots, R–1$, следовательно, размер сигнатуры g_l много меньше размера КТ ($S(g_l) << S(K_l)$). Поэтому g_l может быть размещена в локальной памяти ВУ.

2. Алгоритм дельта-сжатия FHash

Алгоритм, названный нами FHash (Fixed Hash size algorithm), может применяться как для создания инкрементных, так и дифференциальных КТ и отличается высокой скоростью работы. Для вычисления сигнатуры g_l контрольная точка K_l разбивается на $R =]S(K_l) / d[$ блоков длиной d байт $S(B_{lj}) = d$, $\forall j \in \{1, \ldots, R - 1\}$ за исключением последнего, размер которого определяется по формуле $S(B_{lR}) = S(K_l) - d \cdot (R - 1)$, при этом $Y =]X[$ – ближайшее к X целое, такое, что $Y \geq X$. Для вычисления сжатой КТ C_l, выбирается сигнатура g', соответствующая КТ, относительно которой выполняется сжатие (для ИКТ $g' = g_{l-1}$, для ДКТ $g' = g_0$). Затем по формуле (2) вычисляется сигнатура g_l и, на основе сравнения g_l и g', определяются блоки, которые должны быть сохранены в формируемой инкрементной или дифференциальной КТ:

$$C_l = <V_l, p_l>; \quad (3)$$

$$V_l = \{B_{lj} \mid j \in \{1, \ldots, R\}, g_{li} \neq g'_{li}\}; \quad (4)$$

$$p_l = \{(j, h(B_{li})) \mid j \in \{1, \ldots, R\}, g_{li} \neq g'_{li}\}, \quad (5)$$

где V_l – множество блоков K_l, модифицированных относительно КТ, соответствующей сигнатуре g', а p_l – сигнатура, описывающая блоки из V_l. В качестве h выбирается «сильная» хеш-функция с малой вероятностью коллизии (например, MD4, MD5, SHA1). Недостатком данного алгоритма является то, что он не способен обнаруживать вставку и удаление фрагментов КТ.

Получена оценка вычислительной сложности алгоритма FHash. Она складывается из сложностей вычисления хеш-функции для каждого блока (для MD4, MD5, SHA1 она линейна) и сравнения сигнатур g_l и g': $T_{\text{FHash}} = O(Rd) + O(n) = O(n)$. Полученные результаты подтверждаются экспериментальными данными (рис. 2, *а*).

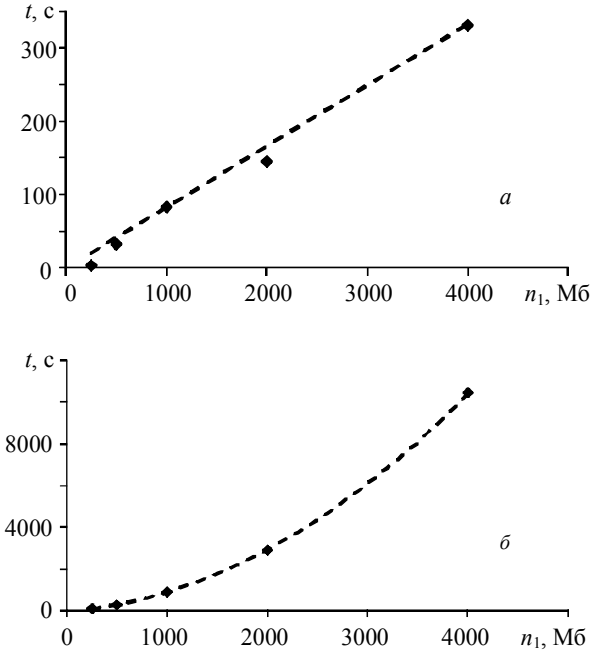

Рис. 2. Время выполнения дельта-сжатия для КТ объемом n и суммарным объемом n_1 модифицированных фрагментов ($n = n_1$): *а* – алгоритмом FHash; *б* – алгоритмом SHash; ◆ – экспериментальное время дельта-сжатия; **– – –** – приближение в соответствии с теоретической оценкой

3. Алгоритмы скользящего окна

Существуют алгоритмы, способные обнаруживать все виды изменений одной КТ относительно другой, они используют принцип «скользящего окна». Рассмотрим базовый алгоритм, названный нами SHash (Sliding window Hash-based algorithm). Сигнатура КТ K_l формируется из пар вида $g_{lj} = \langle w_{lj}, s_{lj} \rangle = h(B_{lj})$, $j = 1, \ldots, R$, где w_{lj}, s_{lj} – это хеш-коды, вычисленные для блока B_{lj}, размер которого полагается фиксированным и равным d. При этом w_{lj} – это так называемый «слабый» признак, для вычисления которого используется хеш-функция h', не требующая значительных вычислительных затрат, а s_{lj} – это «сильный» признак, для вычисления которого, используется хеш-функция h'' с малой вероятностью коллизии. Пусть рассматривается блок $Q_0 = (K_{li}, \ldots, K_{l,(i+d)})$. При этом предыдущий блок $(K_{l,(i-d)}, \ldots, K_{l,(i-1)})$ был обнаружен в сигнатуре g' под номером $j - 1$, тогда номер текущего предполагается равным j. Далее вычисляется пара хеш-кодов для этого блока $\langle w_0, s_0 \rangle = h(Q_0)$, если она совпадает с парой $\langle w'_j, s'_j \rangle$ из эталонной сигнатуры g', то блок не модифицирован. Если пары отличаются, то формируется набор блоков $Q_f = (K_{l,(i+f)}, \ldots, K_{l,(i+f+d)})$, $\forall f = 1, \ldots, (S(K_l) - (i + d))$, для каждого из которых вы-

числяется $w_f = h'(Q_f)$ и выполняется поиск пары $<w_{lj'}, s_{lj'}> \in g'$, такой, что $w_{lj'} = w_f$. Если пара найдена, то выполняется проверка равенства сильных признаков $s_{lj'} = h''(Q_f)$. Если это условие выполняется, то набор байт (K_{li}, …, $K_{l,(i+f-1)}$) считается модифицированным, набор ($K_{l,(i+f)}$, …, $K_{l,(i+f+d-1)}$) считается обнаруженным блоком с номером j'. Работа алгоритма продолжается, исходя из предположения, что блок ($K_{l,(i+f+d)}$, …, $K_{l,(i+f+2\cdot d-1)}$) имеет номер ($j' + 1$).

Для алгоритма SHash также была получена оценка вычислительной сложности, которая составила $T_{\text{SHash}} = O(n_1^2) + O(n - n_1)$, где $n = S(K_l)$, n_1 – суммарный объем измененных фрагментов. Экспериментальные данные (рис. 2, *б*) подтверждают теоретические оценки. В качестве реализации алгоритма «скользящего окна» использовался инструментарий синхронизации данных rsync [13].

Для современных параллельных программ не характерно интенсивное использование динамической памяти, поэтому вставка или удаление фрагментов в КТ встречаются реже, чем их модификация. С другой стороны, накладные расходы на обнаружение измененных фрагментов являются критичными в задаче снижения времени сохранения КТ. В связи с этим алгоритмы, основанные на принципе «скользящего окна», не пригодны для дельта-сжатия КТ.

4. Адаптивный подход к дельта-сжатию контрольных точек

В работе предложен адаптивный подход к дельта-оптимизации КТ, сочетающий в себе преимущества инкрементного и дифференциального дельта-сжатия. Суть подхода состоит в следующем: по умолчанию создаются дифференциальные КТ; если разность между объемами модифицированных данных относительно базовой и предыдущей контрольной точек превышает некоторое пороговое значение, то инициируется создание инкрементной КТ, которая становится базовой для последующих дифференциальных КТ. Для дельта-сжатия КТ K_l требуется две сигнатуры: 1) g_b, описывающая базовую КТ, где $b \in \{1, \ldots, l\}$ – номер текущей базовой КТ (изначально $b = 1$); 2) g_{l-1}, описывающая предыдущую КТ K_{l-1}.

Были проведены исследования предложенного подхода для параллельной программы численного моделирования газодинамических процессов в камере сгорания автомобильного устройства безопасности (Airbag) [14]. Они показали (рис. 3), что размер адаптивной КТ сопоставим с размером инкрементной КТ. И при этом для формирования результирующей КТ требуется: 1) при $l \le 6$ – базовая КТ K_1 и дифференциальная КТ C_l; 2) при $7 \le l \le 25$ базовая КТ K_1, инкрементная КТ C_7 и (при $l > 7$) дифференциальная КТ C_l.

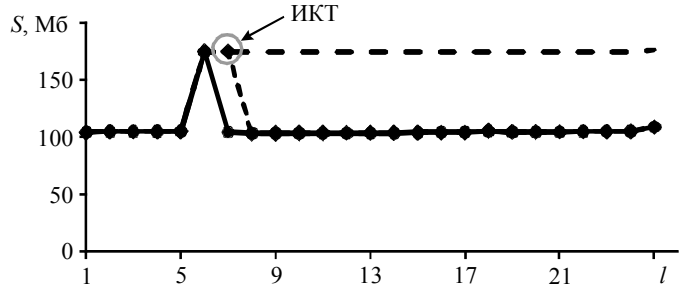

Рис. 3. Эффективность адаптивного подхода к дельта-сжатию: S – размер КТ в Мб; l – номер КТ; —●— – инкрементная КТ; – – – – дифференциальная КТ; –●– – адаптивная КТ

5. Эффективность алгоритмов дельта-сжатия контрольных точек восстановления для параллельных программ

Рассмотренный алгоритм FHash и предложенный адаптивный подход к дельта-оптимизации КТ легли в основу универсального программного инструментария дельта-оптимизации КТ HBICT (Hash Based Incremental Checkpointing Tool) [15]. В пакете HBICT предусмотрено два режима дельта-сжатия контрольных точек: контролируемый и автоматический. Контролируемый режим позволяет: 1) создавать базовую КТ K_b и ее сигнатуру g_b; 2) создавать для некоторой КТ инкрементную или дифференциальную относительно заданной сигнатуры g'; 3) обновлять текущую сигнатуру g' до g_l для заданной КТ K_l. Вычисление порядкового номера КТ, хранение сигнатур и управление ими и другой служебной информацией осуществляется внешними программными средствами.

В автоматическом режиме работа со служебной информацией выполняется средствами HBICT. Для проведения натурных экспериментов пакет HBICT был интегрирован со средством создания КТ DMTCP. Также предусмотрена возможность дополнительного сжатия КТ программой GZIP.

Исследования эффективности предложенных алгоритмов выполнялись на пространственно-распределенной мультикластерной вычислительной системе Центра параллельных вычислительных технологий Государственного образовательного учреждения высшего профессионального образования «Сибирский государственный университет телекоммуникаций и информатики» (ЦПВТ ГОУ ВПО «СибГУТИ») и Института физики полупроводников им. А.В. Ржанова Сибирского отделения Российской академии наук (ИФП СО РАН) [16]. Рассматривалось хранилище данных, организованное на базе файловой системы NFS. Все результаты были получены с использованием предложенного адаптивного подхода к дельта-оптимизации КТ.

В качестве исследуемых параллельных программ были выбраны:

1. Программа Airbag [14].

2. Программа CG из промышленного теста NAS Parallel Benchmarks [17], реализующая вычисление системы линейных алгебраических уравнений методом сопряженных градиентов.

На рис. 4 приведены результаты моделирования различных алгоритмов сохранения КТ для обеих программ, выполнявшихся на 8 вычислительных узлах. Видно, что дельта-сжатие является эффективным и позволяет значительно сократить объем КТ (рис. 4, *а*, *в*). Для программы Airbag сжатие, реализованное в программе GZIP, позволяет уменьшить этот объем практически втрое. Наиболее эффективным является сочетание двух алгоритмов сжатия. Очевидно, что время создания КТ зависит от ее объема (рис. 4, *б*), поэтому наилучший результат показывает комбинированное сжатие.

Для программы CG объем КТ, полученной при дельта-сжатии, уменьшается до 0,1 Гб (рис. 4, *в*), поэтому использование дополнительного сжатия с помощью программы GZIP нецелесообразно. Очевидно, что время создания КТ с использованием дельта-сжатия для этой программы также наилучшее (рис. 4, *г*).

Таким образом, для программ Airbag и CG технология дельта-сжатия является эффективной. При комбинировании дельта-сжатия с другими видами сжатия возможно получение многократного уменьшения объема КТ и времени ее создания.

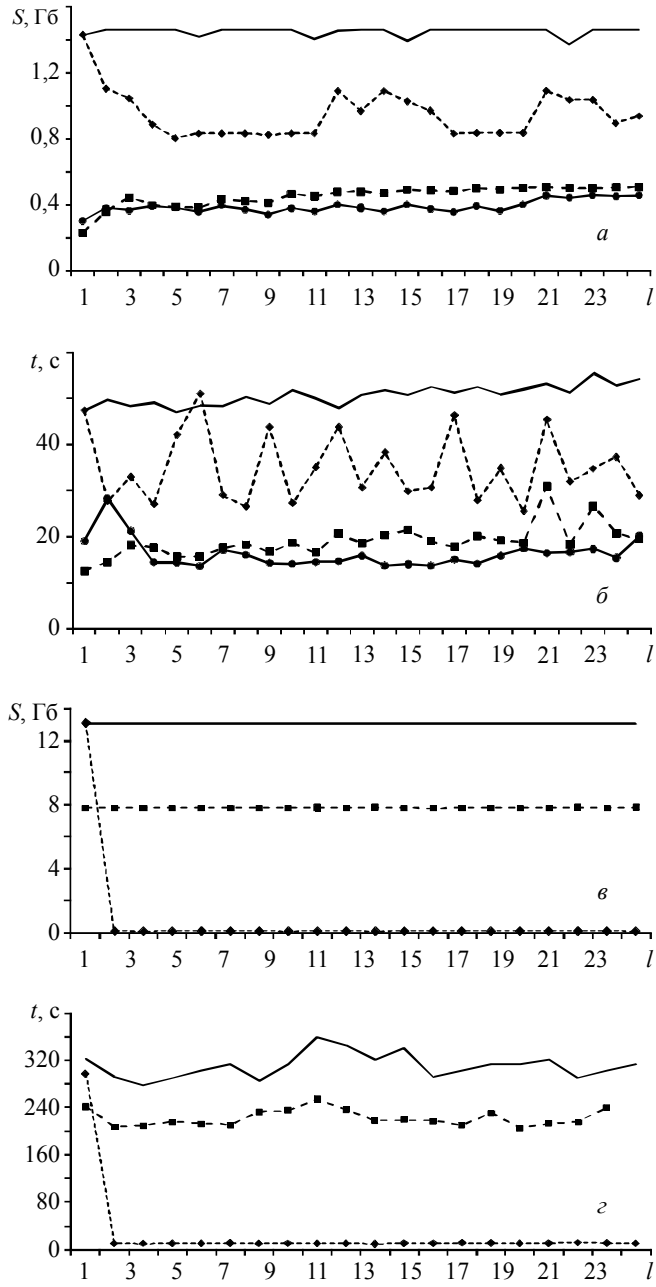

Рис. 4. Накладные расходы на создание распределенной КТ: *а* – размер КТ в Гб (для программы Airbag); *б* – время создания КТ (для программы Airbag); *в* – размер КТ в Гб (для программы CG); *г* – время создания КТ (для программы CG); —— – без применения сжатия; --◆-- – дельта-сжатие HBICT; – ■ – – сжатие программой GZIP; —●— – сжатие HBICT+GZIP

Заключение

В работе предложен адаптивный подход, сочетающий преимущества известных видов дельта-сжатия: инкрементного и дифференциального. Показана его эффективность для набора прикладных программ. Установлено, что использование адаптивного дельта-сжатия позволяет добиться значительного уменьшения объема КТ.

Предложенный подход реализован в программном пакете HBICT, интегрированном со средством создания КТ DMTCP. Разработанный программный пакет является частью системного программного обеспечения пространственно-распределенной мультикластерной вычислительной системы ЦПВТ ГОУ ВПО «СибГУТИ» и ИФП СО РАН.

ЛИТЕРАТУРА

1. *Хорошевский В.Г.* Архитектура вычислительных систем. М.: МГТУ им. Н.Э. Баумана, 2008. 520 с.
2. *TOP500 Supercomputer sites*. URL: http://www.top500.org/ (дата обращения: 18.11.2010).
3. *Philp I.* Software failures and the road to a petaflop machine // Proc. of the Workshop on High Performance Computing Reliability. IEEE Computer Society. 2005.
4. *Team T.B.* An overview of the BlueGene/L supercomputer // Proc. of SC2002: High Performance Networking and Computing. – Baltimore, MD, Nov. 2002.
5. *Budnik T. et al.* High throughput computing on IBM's Blue Gene® // IBM Rochester Blue Gene Development. URL: http://www-03.ibm.com/systems/resources/HTC_WhitePaper_V2_050508.pdf (дата обращения: 18.11.2010).
6. *Elnozahy E.N. et al.* A survey of rollback-recovery protocols in message-passing systems // ACM Computing Surveys. 2002. V. 34. No. 3. P. 375–408.
7. *Hursey J. et al.* The design and implementation of checkpoint/restart process fault tolerance for Open MPI // Proc. of the 21st IEEE International Parallel and Distributed Processing Symposium (IPDPS). IEEE Computer Society, 2007. P. 1–8.
8. *Bosilca G. et. al.* MPICH-V: Toward a scalable fault tolerant MPI for volatile nodes // ACM/IEEE 2002 Conference on Supercomputing. IEEE Press, 2002. P. 29.
9. *Ansel J. et. al.* DMTCP: Transparent checkpointing for cluster computations and the desktop // Proc. of IEEE International Parallel and Distributed Processing Symposium (IPDPS'09). IEEE Press, 2009. P. 1–12.
10. *Agarwal S. et al.* Adaptive incremental checkpointing for massively parallel systems // ICS'04: Proc. of the 18th Annual International Conference on Supercomputing. N.Y.: ACM Press, 2004. P. 277–286.
11. *Kiswany S.A. et al.* stdchk: A checkpoint storage system for desktop grid computing // Proc. of ICDCS. 2008. P. 613–624.
12. *Sangho Yi S. et al.* Adaptive page-level incremental checkpointing based on expected recovery time // Proc. of the 2006 ACM symposium on applied computing. NewYork: ACM, 2006. P. 1472–1476.
13. *Tridgell A.* Efficient algorithms for sorting and synchronization // PhD thesis. The Austrailian National University, 1999.
14. *Рычков А.Д.*, *Шокина Н.Ю.*, *Милошевич Х.* Моделирование процесса зажигания гранулированного унитарного твердого топлива в камере сгорания айрбэга // Материалы междунар. конф. «Вычислительные и информационные технологии в науке, технике и образовании». Павлодар, 2006. Т. 2. С. 165–175.
15. *HBICT* (Hash Based Incremental Checkpointing Tool). URL: http://sourceforge.net/projects/hbict/ (дата обращения: 18.11.2010).
16. *Центр* параллельных вычислительных технологий ГОУ ВПО «СибГУТИ» и ИФП СО РАН. URL: http://cpct.sibsutis.ru/ (дата обращения: 18.11.2010).

17. *NAS* Parallel Benchmarks. URL: http://www.nas.nasa.gov/Resources/Software/npb.html (дата обращения: 18.11.2010).

Поляков Артем Юрьевич
Молдованова Ольга Владимировна
ГОУ ВПО «Сибирский государственный университет
телекоммуникаций и информатики»
E-mail: artpol84@gmail.com; ovmold@yandex.ru

Поступила в редакцию 3 декабря 2010 г.