УДК 004.052.3 DOI: 10.14529/cmse190205

КООРДИНИРОВАННОЕ СОХРАНЕНИЕ С ЖУРНАЛИРОВАНИЕМ ПЕРЕДАВАЕМЫХ ДАННЫХ И АСИНХРОННОЕ ВОССТАНОВЛЕНИЕ В СЛУЧАЕ ОТКАЗА*

© 2019 А.А. Бондаренко, П.А. Ляхов, М.В. Якобовский
Институт прикладной математики им. М.В. Келдыша Российской академии наук (125047 Москва, Миусская пл., д. 4)
E-mail: bondaleksey@gmail.com, pavel.lyakhov@phystech.edu, lira@imamod.ru
Поступила в редакцию: 20.11.2018

Увеличивающийся рост числа компонент суперкомпьютеров приводит специалистов в области HPC к неблагоприятным оценкам для будущих суперкомпьютеров: диапазон среднего времени между отказами будет составлять от 1 часа до 9 часов. Данная оценка ставит под вопрос возможность проведения длительных расчетов на суперкомпьютерах. В работе предлагается метод восстановления после отказов, не требующий возврата большинства процессов к последней контрольной точке, что может позволить сократить накладные расходы для некоторых вычислительных алгоритмов. Стандартный метод обеспечения отказоустойчивости заключается в координированном сохранении, а в случае отказа осуществляется возврат всех процессов к последней контрольной точке. Предлагаемая стратегия заключается в координированном сохранении и журналировании передаваемых данных, а в случае отказа происходит асинхронное восстановление. При асинхронном восстановлении несколько запасных процессов проводят пересчет данных потерянных после отказа, а остальные процессы находятся в ожидании окончания процедуры восстановления потерянных данных. Разработаны параллельные программы, решающие задачу о распространении тепла в тонкой пластине. В данных программах отказы происходят после вызова функции raise (SIGKILL), а координированное или асинхронное восстановление осуществляется с помощью функционала ULFM. Для получения теоретических оценок накладных расходов предложен имитационный метод, моделирующий исполнение программы с отказами. В данном методе отказ может произойти во время расчетов, а также во время сохранения контрольных точек или в ходе восстановления. Проведено сравнение методов восстановления при разных значениях частоты отказов для задачи распространения тепла в тонкой пластине, в которой объем данных для журналирования незначителен. Сравнение показало, что применение асинхронного восстановления приводит к сокращению накладных расходов от 22 % до 40 % при теоретической оценке и от 13 % до 53 % в вычислительном эксперименте.

Ключевые слова: MPI, расширение ULFM, контрольные точки, координированное сохранение, асинхронное восстановление, отказоустойчивость.

ОБРАЗЕЦ ЦИТИРОВАНИЯ

Бондаренко А.А., Ляхов П.А., Якобовский М.В. Координированное сохранение с журналированием передаваемых данных и асинхронное восстановление в случае отказа // Вестник ЮУрГУ. Серия: Вычислительная математика и информатика. 2019. Т. 8, № 2. С. 76–91. DOI: 10.14529/cmse190205.

Введение

В начале двухтысячных годов проходило масштабное наблюдение за отказами в вычислительных системах Лос-Аламоса. В нем было задействовано 22 кластера и около 5000

*Работа рекомендована программным комитетом международной конференции «Суперкомпьютерные дни в России 2018»

узлов. Результаты исследования показали, что за год в среднем в пересчете на один процессор приходится от 0,1 до 0,25 отказа в системе [1]. Наблюдения за использованием вычислительных систем Терафлопсного уровня показали, что время между отказами/прерываниями измеряется десятками часов; например, для некоторых машин эта величина была между 6,5 и 40 часами [2]. Работа с Blue Waters — одним из суперкомпьютеров Петафлопсного уровня [3] — показала, что в среднем каждые 4,2 часа происходили сбои, которые требовали локального исправления, а каждые 160 часов происходили сбои, требующие корректирования работы всего компьютера в целом. Современные суперкомпьютеры сложно устроены и включают в себя сотни тысяч, а некоторые даже миллионы ядер, десятки тысяч процессоров и тысячи узлов и, как правило, различные ускорители. Поэтому прогнозы специалистов для будущих суперкомпьютеров не утешительны, даже если удастся увеличивать время бесперебойной работы для составляющих компонентов, в целом для суперкомпьютеров среднее время между отказами будет между 1 и 9 часами [4].

Поэтому для проведения высокопроизводительных вычислений представляется важным решение следующей задачи: разработать принципы сохранения контрольных точек за время, меньшее характерной продолжительности безотказной работы системы, и алгоритмы, обеспечивающие, в случае отказа части оборудования, быстрое автоматическое возобновление расчета на работоспособной части вычислительного поля. В данной работе рассматривается модификация многоуровневого метода обеспечения отказоустойчивости, которая заключается в координированном сохранении и журналировании передаваемых данных, а в случае отказа происходит асинхронное восстановление. Предлагаемая стратегия позволяет сократить накладные расходы на обеспечение отказоустойчивости расчетов для некоторых вычислительных алгоритмов.

В первом разделе изложен многоуровневый метод обеспечения отказоустойчивости и предлагается модификация метода восстановления. Во втором разделе приводятся: особенности реализации параллельной программы, которая решает задачу о распределении тепла в тонкой пластине; имитационный метод, моделирующий исполнение программы с отказами, который позволяет получить теоретические оценки накладных расходов. В заключительном разделе приводятся результаты вычислительных экспериментов, а именно оценки накладных расходов для разных стратегий восстановления.

1. Стратегии сохранения и восстановления после отказов для многоуровневого метода обеспечения отказоустойчивости

Первый раздел посвящен методам обеспечения отказоустойчивых вычислений. Применение существующих средств сохранения контрольных точек на системном уровне, основанных на BLCR [5], показало ограниченность данного подхода для больших систем [6]. Наиболее перспективным считается подход, заключающийся в передаче пользователю функций по обработке отказа, так как пользователь сможет существенно сократить объем контрольных точек. Одним из таких методов является многоуровневый метод, который также позволяет применять разные стратегии отказоустойчивости в зависимости от информации об отказах в системе.

1.1. Многоуровневый метод обеспечения отказоустойчивости

Многоуровневый метод позволяет объединить разные алгоритмы [4, 6–10] обеспечения отказоустойчивости, в частности, разнообразные методы сохранения контрольных точек, что позволяет адаптироваться под различные типы отказов. Как правило, каждый уровень соответствует конкретному типу отказа и связан со стратегией хранения, которая позволяет провести восстановление после этого типа отказа.

При описании многоуровневого метода [11] принимаются следующие допущения:

– в системе могут происходить отказ от 1 до k уровней;
– наступление отказов разных уровней — независимые случайные величины;
– отказ j-го уровня имеет свою частоту отказа λ_j, свои накладные расходы на сохранение контрольной точки C_j и на восстановление после отказа R_j;
– отказ j-го уровня уничтожает все контрольные точки на нижних уровнях от 1 до j–1 и приводит к необходимости восстановления с уровня j или выше;
– восстановление после отказа j-го уровня должно восстановить данные контрольных точек всех нижних уровней;
– частота отказов убывает, а накладные расходы на сохранение и восстановление возрастают с ростом уровня, то есть $\lambda_1 \geq \lambda_2 \geq ... \geq \lambda_k$, $C_1 \leq C_2 \leq ... \leq C_k$ и $R_1 \leq R_2 \leq ... \leq R_k$.

Замечание 1. Ниже будет использоваться следующие выражения: «отказ j-го уровня», «сохранение j-го уровня», «восстановление j-го уровня». Под данными выражениями следует понимать, что в многоуровневом методе выбран набор отказов для каждого уровня, а также были определены стратегии сохранения и восстановления для соответствующего отказа, которые удовлетворяют допущениям описанным выше.

Замечание 2. Важной задачей применения многоуровневых методов является задача определения оптимальных периодов сохранения контрольных точек для каждого уровня. Один из подходов к решению данной проблемы заключается в составлении специальных шаблонов сохранения контрольных точек всех уровней на заданном участке времени и последующее тиражирование данного шаблона. В данной работе вопросы оптимизации периодов сохранения и составления шаблона не рассматриваются. Более подробно с данными вопросами можно ознакомиться в работах [12, 13].

Одним из примеров многоуровневых методов обеспечения отказоустойчивости служит трехуровневый метод, в котором авторы работы [12] выбрали следующие уровни отказов:

– отказ первого уровня — временные ошибки в памяти;
– отказ второго уровня — сбои в узлах, которые делают недоступными данные из оперативной памяти;
– отказ третьего уровня — сбои, приводящие к недоступности данных из оперативной памяти и данные из резервных копий соседних узлов.

Сохранение ключевых данных для первого уровня будет осуществляться в оперативную память, для второго уровня осуществляется дублирование данных некоторым соседям, для третьего уровня сохранение осуществляется в распределенную файловую систему.

1.2. Координированное сохранение и восстановление для многоуровневого метода

Координированное сохранение подразумевает сначала остановку расчетов и связанных с ними коммуникационных функций, и последующее копирование контрольных точек на выбранные устройства хранения. Каждый многоуровневый метод требует выбора своего набора стратегий сохранения данных исходя из параметров: наличие в вычислительной системе локальных (оперативная память, SSD, HDD) и глобальных средств хранения; скорости чтения/записи данных; объема контрольных точек; информации о частоте отказов. Поэтому выбор шаблона сохранения будет зависеть от применяемого вычислительного алгоритма и характеристик вычислительной системы.

На рис. 1 представлен шаблон сохранения контрольных точек для трех уровней, где по оси абсцисс время исполнения программы, по оси ординат уровни сохранения, черными квадратами отмечены моменты сохранения контрольных точек j-го уровня, P_j — периоды сохранения для соответствующего уровня. Для упрощения представления на рис. 1 время сохранения контрольных точек равно нулю.

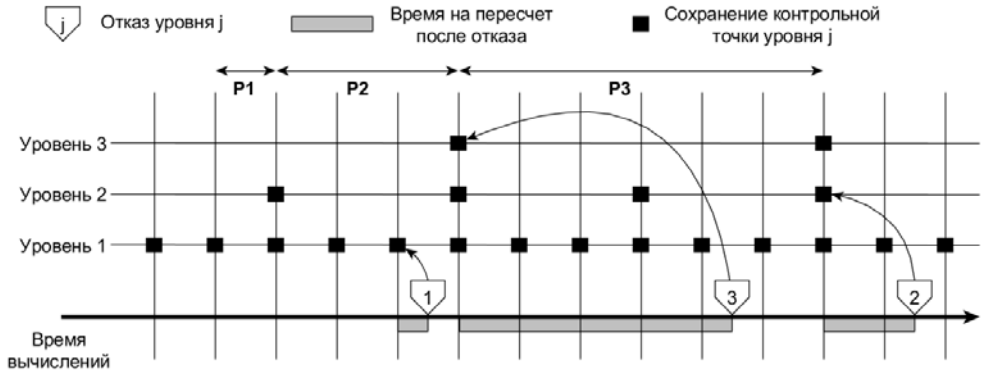


Рис. 1. Трехуровневый шаблон сохранения контрольных точек

В случае обнаружения отказа в системе необходимо осуществить запуск процедуры восстановления, которая описана на рис. 2. Первый шаг позволяет восстановить MPI среду для продолжения вызова коммуникационных функций. Если можно провести декомпозицию всех данных из последней контрольной точки на множество процессов избежавших отказа, то выполнять второй шаг не надо. Однако в данной работе предполагается, что такая декомпозиция не проводится и отказавшие процессы замещаются процессами, которые не принимали участие в расчете с самого начала, то есть были запасными. После второго шага часть процессов, принимавшая участие в расчете и избежавшая отказа, будет помнить состояние (расчетные данные) в момент отказа, но новый процесс, вновь введенный вместо отказавшего процесса, будет иметь состояние (расчетные данные) из последней контрольной точки. Для продолжения расчета необходимо осуществить синхронизацию состояний. Либо состояния расчетных процессов избежавших отказа вернуть к состояниям из последней контрольной точки, либо состояние нового процесса довести до состояния расчетных процессов избежавших отказа. Под координированным восстановлением понимают, возврат всех процессов к состояниям из последней контрольной точки, в этом и состоит третий шаг, описанный на рис. 2. Четвертый шаг связан с окончанием процедуры восстановления и возвратом к нормальному ходу вычислительного процесса.

Шаг 1. Перенастройка MPI среды для нормальной работы коммуникационных функций.
Шаг 2. Взамен отказавшим процессам для проведения расчетов вводятся резервные процессы (или новые процессы, если возможна генерация процессов), таким образом, что новые процессы имеют номера отказавших процессов.
Шаг 3. Теперь для возобновления работы все процессы должны осуществить чтение данных из последней контрольной точки.
Шаг 4. Осуществляется запуск продолжения расчетов.

Рис. 2. Алгоритм координированного восстановления после отказа

На рис. 1 приведены возможные отказы разных уровней, для которых стрелки указывают на контрольные точки, из которых необходимо будет производить координированное восстановление. Серым цветом на рис. 1 указано время потерянных вычислений, то есть время с последней контрольной точки восстановления до момента отказа. При координированном восстановлении, после возврата к контрольной точке, происходит повторный пересчет времени указанного серым цветом.

1.3. Координированное сохранение с журналированием и асинхронное восстановление для многоуровневого метода

Основная цель данной работы заключается в уменьшении накладных расходов расчетов, которые проходят на вычислительной системе с частыми отказами. Представляет интерес вопрос: «Как влияет синхронизация состояний процессов в процедуре восстановления на общий объем накладных расходов?» Было решено исследовать переход на асинхронное восстановление, когда синхронизация осуществляется за счет того, что состояние нового процесса доводят до состояния расчетных процессов избежавших отказа. Для этого предлагается использовать несколько запасных процессов, которые должны осуществить пересчет потерянного времени («серая область» на рис. 1).

Отметим, что с момента сохранения последней контрольной точки до наступления отказа, процессы в общем случае будут обмениваться информацией. Чтобы запасные процессы могли выполнить пересчет потерянных данных им будут нужна информация о том «когда», «какой процесс», «какие данные», предавал отказавшему процессу в период между последним сохранением до наступления отказа. Сохранение этой информации обозначим как журналирование (logging) передаваемых данных. Журналирование требует от каждого процесса сохранения: момента передачи, передаваемых данных и адресата передачи.

В рамках данной работы рассматривается асинхронное восстановление только для отказа первого уровня, что позволяет ограничить область хранения журнальных данных до периода сохранения контрольных точек на первом уровне. Таким образом, должно осуществляться координированное сохранение контрольных точек согласно выбранному шаблону, при этом каждый процесс обязан журналировать передаваемые им данные в течении P_1. На следующем периоде, журнальные данные с предыдущего периода не нужны и они могут быть удалены или перезаписаны.

Для реализации асинхронности в процедуре восстановления (рис. 2) необходимо заменить третий шаг (рис. 3).

Шаг 3. Процессы, избежавшие отказа, делятся на три группы.

- Первая группа – процессы, которые с момента последнего сохранения контрольной точки осуществляли передачу данных отказавшему процессу.
- Вторая группа – резервные процессы, которые будут осуществлять пересчет данных, потерянных в результате отказа.
- Третья группа – остальные процессы, которые будут ожидать окончание процедуры восстановления.

Процессы первой группы осуществляют передачу журнальных данных процессам из второй группы. Процессы второй группы осуществляют пересчет потерянных расчетов с последней контрольной точки до момента отказа и принимают необходимые журнальные данные.

Рис. 3. Шаг 3 алгоритма асинхронного восстановления после отказа

Отметим, что на пересчет выделяются резервные процессы в кратном размере, чтобы ускорить процесс восстановления по сравнению с восстановлением за счет одновременного возврата всех процессов к последней контрольной точке.

Замечание 3. Пусть $V_{жур}$ — объем передаваемых данных одним процессом за вычислительную итерацию, L — количество вычислительных итераций в течении P_1, то есть между сохранениями контрольных точек на первом уровне. Если произведение $V_{жур}{\cdot}L$ достаточно мало, чтобы поместиться в оперативную память вместе с другими данными для расчетов, то можно осуществлять журналирование целиком в оперативную память вычислительных узлов. Таким образом, время на журналирование несущественно и его можно считать равным нулю. Однако если $V_{жур}{\cdot}L$ имеет существенный объем, то потребуется разбить L на части для периодической записи журналов передаваемых данных на локальные устройства хранения (SSD/HDD), что приведет к необходимости оценивать соответствующие накладные расходы. В рамках данной работы время на журналирование считается равным нулю, а оценка накладных расходов на журналирование для различных вычислительных алгоритмов остается задачей для будущих работ.

2. Оценка накладных расходов

Для сравнения методов обеспечения отказоустойчивости, описанных в первом разделе, реализованы параллельные программы, решающие тестовую задачу. В программах отказ является случайной величиной, принадлежащий экспоненциальному распределению, который реализуются вызовом функции raise (SIGKILL). Обработка отказа (рис. 2) происходит с помощью функционала ULFM. Однако такой подход не позволяет получить большую выборку накладных расходов, поэтому в работе предлагается имитационный метод, моделирующий исполнение программы, для оценки накладных расходов. Особенность метода состоит в том, что отказ может произойти во время расчетов, а также во время сохранения контрольных точек или в ходе восстановления. Данный метод позволяет моделировать многократное исполнение прикладной программы со случайными отказами.

2.1. Программная реализация решения краевой задачи распределения тепла в тонкой прямоугольной пластине с отказами процессов

В данной работе для вычислительного эксперимента выбрана краевая задача распределения тепла в тонкой прямоугольной пластине. Для проведения вычислений применяется явная разностная схема и геометрический метод для параллельной реализации, так чтобы каждый процесс получал одинаковое количество узлов начальной сетки. Общий объем сетки составляет 256 миллионов узлов. Итерационный процесс вычисления сопровождается измерениями времени, которое играет ключевую роль для определения наступления события: сохранения контрольной точки в оперативную память или в распределенную файловую систему, а также наступление отказа. Вычисления продолжается до тех пор, пока не будет выполнен заранее запланированный объем расчетов ($T_з$). Общий алгоритм программ представлен на рис. 4. Наличие отказа в системе не мешает автономной работе каждого из процессов, а детекция отказа происходит только при выполнении коммуникационных функций, поэтому в данной работе запуск процедуры восстановления осуществляется только на третьем, пятом и седьмом шагах.

Выбраны следующие отказы: отказ первого уровня — незначительные сбои в системе, для которых достаточно данных, хранящихся в оперативной памяти соседнего процесса, отказ второго уровня — более сложные сбои, для восстановления после которых необхо димо хранить данные в распределенной файловой системе. Сохранение контрольных то-

Шаг 1. Каждый процесс проводит свои вычисления.
Шаг 2. Проверяется условие наступления отказа. В случае положительного ответа, один из процессов выполняет функцию raise(SIGKILL).
Шаг 3. Процессы производят обмен данными. Если при выполнении коммуникационных функций обнаружен отказ, то не отказавшие процессы выполняют процедуру восстановления.
Шаг 4. Проверяется условие наступления отказа. В случае положительного ответа, один из процессов выполняет функцию raise(SIGKILL).
Шаг 5. Происходит сохранение контрольных точек при выполнении соответствующих условий. Если в коммуникационных функциях обнаружен отказ, то не отказавшие процессы выполняют процедуру восстановления.
Шаг 6. Проверяется условие наступления отказа. В случае положительного ответа, один из процессов выполняет функцию raise(SIGKILL).
Шаг 7. Проверяется продолжение расчетов и происходит переход на шаг 1. Если при выполнении коммуникационных функций обнаружен отказ, то не отказавшие процессы выполняют процедуру восстановления.

Рис. 4. Алгоритм основной части программы с отказами

чек для первого уровня осуществляется в оперативную память соседнего процесса, а для второго уровня — в распределенную файловую систему.

Разработаны две программы с разными стратегиями обеспечения отказоустойчивости. Первая программа реализует координированное сохранение по шаблону для двух уровней, а в случае отказа восстановление происходит за счет координированного возврата всех процессов к последней контрольной точке соответствующего уровня. Вторая

программа реализует координированное сохранение по шаблону и журналирование передаваемых данных. В случае отказа первого уровня осуществляется асинхронное восстановление с двумя или пятью резервными процессами для пересчета, а в случае отказа второго уровня восстановление происходит за счет координированного возврата всех процессов к последней контрольной точке.

Для реализации многоуровневого метода необходимо определить ключевые параметры метода. Выбираются отказы и соответствующие алгоритмы сохранения и восстановления. Пусть известно среднее время между отказами (MTBF). Для выбранной вычислительной задачи надо определить набор ключевых данных, необходимых для осуществления восстановления в случае отказа. После этого определяются характеристики записи контрольных точек для каждого уровня C_i, а также значения времени восстановления системы и чтения этих данных с соответствующего носителя R_i. Далее необходимо определить параметры шаблона сохранения этих данных, а именно длину шаблона и количество сохранений контрольных точек разных уровней (W^{onm}, N_i^{onm}). Решение этой задачи описано в работе [12] (рис. 5).

```
void opt_par(double *lam, double *N^opt, double *C, int count, double &W^opt){
     double S_bot(0), S_top(0);
     for(int i=0;i<count;i++){
          N^opt [i] = sqrt(C(count-1)*lam(i)/lam(count-1)/C(i));
          S_top = S_top + N^opt (i)*C(i);
          S_bot = S_bot + lam(i)/N^opt(i);
     }
     W^opt = sqrt(2*S_top/S_bot);
}
```

Рис. 5. Процедура определения оптимальных параметров шаблона сохранения

Отказы рассматриваются как случайные величины, описываемые экспоненциальным распределением. Для реализации набора отказов в расчете используется массив времен, заполненный по алгоритму (рис. 6), который осуществляет генерацию случайных величин, следующих друг за другом и подчиняющихся экспоненциальному распределению. Когда время работы программы превышает время следующего отказа, то в системе фиксируется отказ и осуществляется запуск процедуры восстановления.

```
void generate_failures(float *Failures,int count, float lambda){
     double p(0),q(0);
     for(int i=0;i<count;i++){
          q=(double)rand()/(double)RAND_MAX;
          p=p-log(q)/lambda;
          Failures[i]=p;
     }
}
```

Рис. 6. Процедура генерации массива случайных чисел, следующих друг за другом и подчиняющихся экспоненциальному распределению

Стандартная версия MPI (MPI 3.1) не содержит средств обработки отказов при коммуникационных операциях. Для обнаружения отказа в системе, распространения информации об ошибке и последующем восстановлении среды MPI используются функции ULFM [14]: MPIX_Comm_agree(), MPIX_Comm_revoke(), MPIX_Comm_shrink(). Примеры реализации программ с ULFM и описанием его функционала можно найти в [14, 15].

2.2. Имитационный метод, моделирующий исполнение параллельной программы подверженной отказам

Имитационный метод, описывающий исполнение параллельной программы подверженной отказам, учитывает возможность наступления случайных отказов во время сохранения контрольных точек и во время восстановления расчетов, а также позволяет сравнивать различные техники обеспечения отказоустойчивости. Метод заключается в разделении времени исполнения программы на небольшие части величиной Δt и последовательном итерировании времени исполнения программы с шагом Δt. Такие события как «сохранение контрольной точки» или «наступление отказа» происходят, когда время работы программы не меньше времени наступления соответствующего события.

В основе предлагаемой имитационной модели лежат следующие положения.

– Программа моделирования должна оперировать с двумя переменными, описывающими время: $T_{\text{выч}}$ — время вычислений основной части кода, без учета накладных расходов на обеспечение отказоустойчивости, $T_{\text{тек}}$ — текущее время работы программы, включающее накладные расходы на обеспечение отказоустойчивости.
– Моделирование исполнения программы с отказами продолжается пока не будет выполнен весь заранее запланированный расчет $T_{\text{з}}$.
– По текущему времени работы $T_{\text{тек}}$ определяется наступление отказа в системе и сохранение контрольных точек.
– Ход выполнения программы осуществляется с некоторым шагом по времени Δt.

На рис. 7 представлен алгоритм имитационного метода описания исполнения параллельной программы подверженной отказам. Функция get time for saves() определяет, происходит ли на данной итерации запись контрольной точки на каком-либо уровне, и в соответствии с этим возвращает TFS — время на сохранение контрольных точек для текущей итерации. Функция get failure info () определяет FI — уровень и время отказа на данной итерации. Изменение переменных $T_{\text{выч}}$, $T_{\text{тек}}$ происходит с помощью функций get comp difference(), get cur difference().

```
while Tвыч<Tз do
     TFS = get time for saves (Tвыч,Δt,Ci)
     FI = get failure info(Tтек,Δt,TFS)
     Tвыч = Tвыч+get comp difference (Tвыч,Δt,FI)
     Tтек = Tтек+get cur difference (Tвыч,Tтек,Δt,TFS,FI,Ri)
end while
return Tтек
```

Рис. 7. Алгоритм имитационного моделирования программы с отказами

Приведем значения функций get comp difference(), get cur difference() при координированном восстановлении после отказа. Пусть шаг одной итерации по времени будет равен периоду сохранения контрольных точек на первом уровне ($\Delta t = P_1$). В табл. 1 приведены значения этих функций для ситуаций с отказами разных уровней представленных на рис. 8.

При отсутствии отказа на данной итерации, время основных вычислений ($T_{\text{выч}}$) должно увеличиться на значение шага $\Delta t = P_1$, а для текущего времени работы ($T_{\text{тек}}$) необходимо дополнительно учесть накладные расходы на сохранение, то есть

$$\text{get comp difference}() = \Delta t,\ \text{get cur difference}() = \Delta t + TFS.$$

В случае отказа на данной итерации, время основных вычислений ($T_{\text{выч}}$) должно вернуться к значению последней контрольной точки, то есть уменьшиться на значение $x{\cdot}\Delta t$, где x — число потерянных итераций. Текущее время работы ($T_{\text{тек}}$) должно увеличиться на время восстановление системы после отказа и на время вычислений сделанных в рамках этой итерации Y(«серое время» на рис. 8), то есть

get comp difference() = -$x{\cdot}\Delta t$, get cur difference() = R_i + Y.

Таблица 1

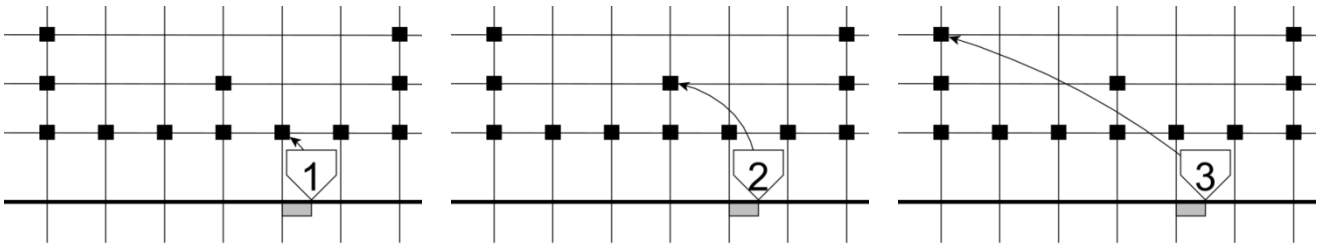
Значения функций при координированном восстановлении

	Нет отказа	Отказ первого уровня	Отказ второго уровня	Отказ третьего уровня
get comp difference()	Δt	0	-Δt	-4Δt
get cur difference()	Δt + *TFS*,	R_i +Y	R_i +Y	R_i +Y

Для получения предварительной оценки «максимального эффекта» от асинхронности восстановления в имитационном методе моделирования предполагается, что применение k процессов для пересчета позволит сократить время самого пересчета в k раз. Данное допущение позволяет получить верхнюю оценку сокращения накладных расходов обеспечения отказоустойчивости от применения асинхронного восстановления.

Замечание 4. В общем случае ускорение пересчета потерянных данных после отказа будет зависеть от многих факторов, а для некоторых вычислительных алгоритмов представлять отдельную задачу из-за сложности самого алгоритма или сложности декомпозиции данных, с которыми работал отказавший процесс.

В алгоритме имитационного метода при асинхронном восстановлении изменятся только значения функций get comp difference(), get cur difference(). Соответствующие примеры для ситуаций из рис. 8 представлены в табл. 2. Шаг одной итерации равен периоду сохранения контрольных точек на первом уровне ($\Delta t = P_1$).



а) отказ первого уровня б) отказ второго уровня в) отказ третьего уровня

Рис. 8. Примеры отказов разных уровней

При отсутствии отказа на данной итерации, расчетное время должно увеличиться на значение шага $\Delta t = P_1$, а в текущее время работы дополнительно надо учесть накладные расходы на сохранение, то есть

get comp difference() = Δt, get cur difference() = Δt + *TFS*.

В случае отказа на конкретной итерации необходимо завершить восстановление, поэтому время основных вычислений на данной итерации не будет увеличиваться. А для текущего времени работы должны быть учтены накладные расходы на восстановление после отказа (R_i) и время на синхронизацию состояний процессов (x·Δt/k+Y/k), так как

потерянным будет время от последней контрольной точки до момента наступления отказа (x·Δt+Y), а при пересчете будут участвовать k процессов.

get comp difference() = Δt, get cur difference() = R_i +(x·Δt+Y)/k.

Таблица 2

Значения функций при асинхронном восстановлении

	Нет отказа	Отказ первого уровня	Отказ второго уровня	Отказ третьего уровня
get comp difference()	Δt	0	0	0
get cur difference()	Δt + *TFS*,	R_i + Y/k	R_i + (Δt+Y)/k	R_i + (4Δt+Y)/k

В рамках данной работы были реализованы алгоритмы имитационного моделирования работы программы с координированным восстановлением и асинхронным восстановлением с двумя ($k = 2$) и пятью ($k = 5$) процессами принимающими участие в пересчете. В этих программах шаг итерации Δt не зависит от периодов сохранения и принимает значение 0,5 секунды.

3. Вычислительный эксперимент

В рамках вычислительного эксперимента были выбраны следующие пары величин среднего времени между отказами (MTBF) для отказа первого и второго уровня [$T_з$/2 10·$T_з$], [$T_з$/4 $T_з$], [$T_з$/10 $T_з$/2], [$T_з$/15 $T_з$/3], [$T_з$/20 $T_з$/4], где $T_з$ — время заранее запланированного счета $T_з = 3600$ с. Для выбранной сетки время сохранения контрольных точек составляет 1 с и 6 с для первого и второго уровня соответственно, время восстановления после отказов первого и второго уровней равны 0,5 с и 4 с.

Для параллельных программ, реализующих вычислительный алгоритм расчета распределения тепла, было проведено 10 вычислительных экспериментов для каждого набора параметров, а для программ, реализующих имитационный метод моделирования, было проведено 10000 экспериментов.

В табл. 3 приведены параметры, описывающие полученные значения накладных расходов: $\bar{t}$ — среднее время накладных расходов, то есть среднее время работы программы с отказами минус время запланированного счета (3600 с); σ — среднее квадратическое отклонение накладных расходов; $\bar{n}_1$, $\bar{n}_2$ — средние значения числа отказов первого и второго уровней; ΔT — разность средних времен накладных расходов для программ с координированным восстановлением и с асинхронным восстановлением; % — отношение ΔT к среднему значению накладных расходов при координированном восстановлении выраженное в процентах; k — число резервных процессов, принимающих участие в пересчете потерянных данных из-за отказа.

Теоретические оценки средних значений накладных расходов, полученные имитационным методом, показывают, что применение асинхронного метода восстановления приводит к сокращению не менее чем на 22 % при двух процессах для пересчета, а при пяти процессах для пересчета не менее чем на 37 %.

Оценки средних значений накладных расходов, полученные в результате работы параллельных программ с отказами, показывают, что применение асинхронного ме-тода вос становления приводит к сокращению от 13 % до 47 % при двух процессах для пере-счета, а при пяти процессах для пересчета от 23 % до 53 %. Значения средних времен между отказами были выбраны в достаточно широком диапазоне, от редких отказов MTBF =

Таблица 3

Оценка накладных расходов для разных методов восстановления

		Координированное	Асинхронное, $k = 2$	Асинхронное, $k = 5$
MTBF = [1800 36000]	ИМ[1]	$\bar{t} = 187$; $\sigma = 130$ $\bar{n}_1 = 2{,}1$; $\bar{n}_2 = 0{,}1$	$\bar{t} = 142$; $\sigma = 72$ $\bar{n}_1 = 2{,}1$; $\bar{n}_2 = 0{,}1$ $\Delta T = 45$ (24 %)	$\bar{t} = 113$; $\sigma = 30$ $\bar{n}_1 = 2{,}1$; $\bar{n}_2 = 0{,}1$ $\Delta T = 74$ (40 %)
	ПП[2]	$\bar{t} = 225$ $\bar{n}_1 = 2$; $\bar{n}_2 = 0{,}1$	$\bar{t} = 119$ $\bar{n}_1 = 2{,}3$; $\bar{n}_2 = 0$ $\Delta T = 106$ (47 %)	$\bar{t} = 106$ $\bar{n}_1 = 1{,}4$; $\bar{n}_2 = 0{,}1$ $\Delta T = 119$ (53 %)
MTBF = [720 3600]	ИМ	$\bar{t} = 432$; $\sigma = 154$ $\bar{n}_1 = 5{,}6$; $\bar{n}_2 = 1{,}1$	$\bar{t} = 331$; $\sigma = 86$ $\bar{n}_1 = 5{,}5$; $\bar{n}_2 = 1{,}1$ $\Delta T = 101$ (23 %)	$\bar{t} = 271$; $\sigma = 44$ $\bar{n}_1 = 5{,}4$; $\bar{n}_2 = 1{,}1$ $\Delta T = 161$ (37 %)
	ПП	$\bar{t} = 438$ $\bar{n}_1 = 6{,}3$; $\bar{n}_2 = 1{,}6$	$\bar{t} = 328$ $\bar{n}_1 = 4{,}9$; $\bar{n}_2 = 0{,}8$ $\Delta T = 110$ (25 %)	$\bar{t} = 302$ $\bar{n}_1 = 6{,}3$; $\bar{n}_2 = 0{,}9$ $\Delta T = 136$ (31 %)
MTBF = [360 1800]	ИМ	$\bar{t} = 638$; $\sigma = 164$ $\bar{n}_1 = 11{,}8$; $\bar{n}_2 = 2{,}4$	$\bar{t} = 488$; $\sigma = 97$ $\bar{n}_1 = 11{,}4$; $\bar{n}_2 = 2{,}3$ $\Delta T = 150$ (24 %)	$\bar{t} = 400$; $\sigma = 53$ $\bar{n}_1 = 11{,}1$; $\bar{n}_2 = 2{,}2$ $\Delta T = 238$ (37 %)
	ПП	$\bar{t} = 630$ $\bar{n}_1 = 12{,}3$; $\bar{n}_2 = 2{,}8$	$\bar{t} = 545$ $\bar{n}_1 = 12{,}7$; $\bar{n}_2 = 2{,}8$ $\Delta T = 85$ (13 %)	$\bar{t} = 487$ $\bar{n}_1 = 11{,}2$; $\bar{n}_2 = 2{,}4$ $\Delta T = 143$ (23 %)
MTBF = [240 1800]	ИМ	$\bar{t} = 812$; $\sigma = 175$ $\bar{n}_1 = 18{,}5$; $\bar{n}_2 = 3{,}7$	$\bar{t} = 626$; $\sigma = 107$ $\bar{n}_1 = 17{,}6$; $\bar{n}_2 = 3{,}5$ $\Delta T = 186$ (23 %)	$\bar{t} = 513$; $\sigma = 60$ $\bar{n}_1 = 17{,}1$; $\bar{n}_2 = 3{,}4$ $\Delta T = 299$ (37 %)
	ПП	$\bar{t} = 794$ $\bar{n}_1 = 18{,}5$; $\bar{n}_2 = 3{,}5$	$\bar{t} = 664$ $\bar{n}_1 = 16{,}4$; $\bar{n}_2 = 2{,}6$ $\Delta T = 130$ (16 %)	$\bar{t} = 603$ $\bar{n}_1 = 18{,}3$; $\bar{n}_2 = 3{,}4$ $\Delta T = 191$ (24 %)
MTBF = [180 900]	ИМ	$\bar{t} = 955$; $\sigma = 184$ $\bar{n}_1 = 25{,}3$; $\bar{n}_2 = 5{,}1$	$\bar{t} = 744$; $\sigma = 116$ $\bar{n}_1 = 24{,}2$; $\bar{n}_2 = 4{,}9$ $\Delta T = 211$ (22 %)	$\bar{t} = 604$; $\sigma = 65$ $\bar{n}_1 = 23{,}3$; $\bar{n}_2 = 4{,}7$ $\Delta T = 351$ (37 %)
	ПП	$\bar{t} = 938$ $\bar{n}_1 = 26$; $\bar{n}_2 = 4{,}6$	$\bar{t} = 762$ $\bar{n}_1 = 23{,}7$; $\bar{n}_2 = 3{,}8$ $\Delta T = 176$ (19 %)	$\bar{t} = 715$ $\bar{n}_1 = 22{,}7$; $\bar{n}_2 = 4{,}3$ $\Delta T = 223$ (24 %)

$[T_з/2 \;\; 10{\cdot}T_з]$, до частых отказов MTBF = $[T_з/20 \;\; T_з/4]$. В вычислительных экспериментах число отказов в среднем доходило до 30 и более отказов за время работы программы. С увеличением частоты отказов явно растет объем накладных расходов и в случае самых частых отказов средние значения накладных расходов для стратегии координированного восстановления в среднем составляли около 26 % от времени заранее запланированного объема расчетов. Однако явно не прослеживается зависимость объема сокращений накладных расходов от частоты отказов при переходе от координированного к асинхронному восстановлению.

[1] ИМ — результаты полученные имитационным методом.

[2] ПП — результаты полученные параллельной программой.

Заключение

В данной работе рассматриваются два метода обеспечения отказоустойчивости расчетов при отказах в вычислительной системе. Первый метод (стандартный) заключается в координированном сохранении контрольных точек, а в случае отказа осуществляется координированный возврат всех процессов к последней контрольной точке. Второй метод основан на координированном сохранении и журналировании передаваемых данных, а в случае отказа происходит асинхронное восстановление. При асинхронном восстановлении несколько запасных процессов проводят пересчет данных, потерянных после отказа, а остальные процессы находятся в ожидании окончания процедуры восстановления потерянных данных.

В работе предложен имитационный метод, моделирующий исполнение параллельной программы подверженной отказам. Данный метод описывает исполнение параллельной программы, учитывая возможность наступления случайных отказов во время выполнения основного алгоритма, сохранения контрольных точек и восстановления расчетов, а также позволяет сравнивать различные методы обеспечения отказоустойчивости.

Для экспериментальной оценки накладных расходов были реализованы параллельные программы, вычисляющие распределение тепла в тонкой пластине с координированным и асинхронным восстановлением. Для теоретической оценки накладных расходов этих методов был использован имитационный метод. Результаты сравнения оценок накладных расходов позволяют говорить о целесообразности применения асинхронной стратегии восстановления для вычислительных алгоритмов с незначительным объемом данных для журналирования, однако возможность ее применения для других алгоритмов требует дальнейших исследований.

Работа выполнена при финансовой поддержке Российского фонда фундаментальных исследований в рамках научного проекта № 17-07-01604 а.

Литература

1. Schroeder B., Gibson G.A. Understanding Failures in Petascale Computers // Journal of Physics: Conference Series. 2007. Vol. 78, No. 1 P. 12–22. DOI: 10.1088/1742-6596/78/1/012022
2. Hsu C.-H., Feng W.-C. A Power-aware Run-time System for High-performance Computing // Proceedings of the 2005 ACM/IEEE Conference on Supercomputing (Seattle, WA, USA, November 12 – 18, 2005). IEEE, 2005. P. 1–9. DOI: 10.1109/sc.2005.3
3. Martino C.D., Kalbarczyk Z., Iyer R.K., Baccanico F., Fullop J., Kramer W. Lessons Learned from the Analysis of System Failures at Petascale: The Case of Blue Waters // 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (Atlanta, Georgia, USA, June 23 – 26, 2014). IEEE, 2014. P. 610–621. DOI: 10.1109/dsn.2014.62
4. Dongarra J., Herault T., Robert Y. Fault-tolerance Techniques for High-performance Computing. Springer, Cham, 2015. 320 p. DOI: 10.1007/978-3-319-20943-2
5. Berkeley Lab Checkpoint/Restart (BLCR) for LINUX URL: http://crd.lbl.gov/departments/computer-science/CLaSS/research/BLCR/ (дата обращения: 03.11.2018)

6. Cappello F., Geist A., Gropp W., Kale S., Kramer B., Snir M., Toward Exascale Resilience: 2014 Update // Supercomputing Frontiers and Innovations. 2014. Vol. 1, No. 1. P. 5–28. DOI: 10.14529/jsfi140101
7. Elnozahy E.N. M., Alvisi L., Wang Y.-M., Johnson D. B. A Survey of Rollback-recovery Protocols in Message-passing Systems // ACM Comput. Surv. 2002. Vol. 34, No. 3. P. 375–408. DOI: 10.1145/568522.568525
8. Bouteiller A., Herault T., Bosilca G., Du P., Dongarra J. Algorithm-based Fault Tolerance for Dense Matrix Factorizations, Multiple Failures and Accuracy // ACM Transactions on Parallel Computing. 2015. Vol. 1, No. 2. P. 1–28. DOI: 10.1145/2686892
9. Engelmann C., Vallee G.R., Naughton T., Scott S.L. Proactive Fault Tolerance Using Preemptive Migration // 17th Euromicro International Conference on Parallel, Distributed and Network-based Processing (Weimar, Germany, February 18 – 20, 2009). IEEE, 2009. P. 252–257. DOI: 10.1109/PDP.2009.31.
10. Бондаренко А.А., Якобовский М.В. Обеспечение отказоустойчивости высокопроизводительных вычислений с помощью локальных контрольных точек // Вестник ЮУрГУ. Серия: Вычислительная математика и информатика. 2014. Т. 3, № 3. С. 20–36 DOI: 10.14529/cmse140302
11. Di S., Bouguerra M.S., Bautista-Gomez L., Cappello F. Optimization of Multi-level Checkpoint Model for Large Scale HPC Applications // 28th International Parallel and Distributed Processing Symposium (Phoenix, Arizona, USA, May 19 – 23, 2014). IEEE, 2014. P. 1181–1190. DOI: 10.1109/IPDPS.2014.122.
12. Benoit A., Cavelan A., Le Fèvre V., Robert Y., Sun H. Towards Optimal Multi-level Checkpointing // IEEE Transactions on Computers. 2016. Vol. 66, No. 7. P. 1212–1226. DOI: 10.1109/TC.2016.2643660.
13. Di S., Robert Y., Vivien F., Cappello F. Toward an Optimal Online Checkpoint Solution under a Two-level HPC Checkpoint Model // IEEE Transactions on Parallel and Distributed Systems. 2016. Vol. 28, No. 1. P. 244–259. DOI: 10.1109/TPDS.2016.2546248.
14. Fault Tolerance Research Hub URL: http://fault-tolerance.org/ (дата обращения: 03.11.2018)
15. Бондаренко А.А., Ляхов П.А., Якобовский М.В. Накладные расходы, связанные с обеспечением отказоустойчивых вычислений при многоуровневом координированном сохранении контрольных точек // Параллельные вычислительные технологии (ПаВТ'2017): Труды международной научной конференции (Казань, 3 – 7 апреля 2017 г.). Челябинск: Издательский центр ЮУрГУ, 2017. С. 262–270.

Бондаренко Алексей Алексеевич, к.ф.-м.н., старший научный сотрудник, Институт прикладной математики им. М.В. Келдыша РАН (Москва, Российская Федерация)

Ляхов Павел Александрович, аспирант, Институт прикладной математики им. М.В. Келдыша РАН (Москва, Российская Федерация)

Якобовский Михаил Владимирович, член-корреспондент РАН, д.ф.-м.н., профессор, заместитель директора по научной работе, Институт прикладной математики им. М.В. Келдыша РАН (Москва, Российская Федерация)

DOI: 10.14529/cmse190205

COORDINATED CHECKPOINTING WITH SENDER-BASED LOGGING AND ASYNCHRONOUS RECOVERY FROM FAILURE

© 2019 A.A. Bondarenko, P.A. Lyakhov, M.V. Yakobovskiy
Keldysh Institute of Applied Mathematics Russian Academy of Sciences
(sq. Miusskaya 4, Moscow, 125047 Russia)
E-mail: bondaleksey@gmail.com, pavel.lyakhov@phystech.edu, lira@imamod.ru
Received: 20.11.2018

The increasing growth in the number of components of supercomputers leads HPC specialists to unfavorable estimates for future supercomputers: "the range of the mean time between failures will be from 1 hour to 9 hours." This estimate leads to the problem of long calculations on supercomputers. In this paper, we propose a recovery method from failure which does not require rollback for all processes. This method can reduce overhead costs for some computational algorithms. The standard fault tolerance method consists of two phases: coordinated checkpointing and rollback of all processes to the last checkpoint in the case of a failure. The proposed method includes coordinated checkpointing with sender-based logging and asynchronous recovery when most processes wait and several processes recalculate the lost data. We developed parallel programs to solve the problem of heat transfer in the thin plate. In these programs, failures occur by calling the function raise(SIGKILL), and coordinated or asynchronous recovery is performed by ULFM functions. In order to obtain theoretical estimates of overhead costs, we propose a simulation model of program execution with failures. This model assumes that failures strike during computations, checkpointing and recovery. We made a comparison of recovery methods with different failure rates for the problem with a small amount of data for logging. The comparison showed that the use of asynchronous recovery results in a reduction of overhead costs by theoretical estimates from 22 % to 40 %, and by computational experiments from 13 % to 53 %.

Keywords: MPI, ULFM extension, coordinated checkpointing, asynchronous recovery, fault tolerance.

FOR CITATION

Bondarenko A.A., Lyakhov P.A., Yakobovskiy M.V. Coordinated Checkpointing with Sender-based Logging and Asynchronous Recovery from Failure. *Bulletin of the South Ural State University. Series: Computational Mathematics and Software Engineering.* 2019. vol. 8, no. 2. pp. 76–91. (in Russian) DOI: 10.14529/cmse190205.

References

1. Schroeder B., Gibson G.A. Understanding Failures in Petascale Computers. *Journal of Physics: Conference Series.* 2007. vol. 78, no. 1. pp. 12–22. DOI: 10.1088/1742-6596/78/1/012022
2. Hsu C.-H., Feng W.-C. A Power-aware Run-time System for High-performance Computing. *Proceedings of the 2005 ACM/IEEE Conference on Supercomputing (Seattle, WA, USA, November 12 – 18, 2005).* IEEE, 2005. pp. 1–9. DOI: 10.1109/sc.2005.3
3. Martino C.D., Kalbarczyk Z., Iyer R.K., Baccanico F., Fullop J., Kramer W. Lessons Learned from the Analysis of System Failures at Petascale: The Case of Blue Waters. *44th*

Annual IEEE/IFIP International Conference on Dependable Systems and Networks (Atlanta, Georgia, USA, June 23 – 26, 2014). IEEE, 2014. pp. 610–621. DOI: 10.1109/dsn.2014.62

4. Dongarra J., Herault T., Robert Y. *Fault-tolerance Techniques for High-performance Computing.* Springer, Cham, 2015. 320 p. DOI: 10.1007/978-3-319-20943-2
5. *Berkeley Lab Checkpoint/Restart (BLCR) for LINUX.* Available at: http://crd.lbl.gov/departments/computer-science/CLaSS/research/BLCR/ (accessed: 03.11.2018)
6. Cappello F., Geist A., Gropp W., Kale S., Kramer B., Snir M., Toward Exascale Resilience: 2014 Update. *Supercomputing Frontiers and Innovations.* 2014. vol. 1, no. 1. pp. 5–28. DOI: 10.14529/jsfi140101
7. Elnozahy E.N. M., Alvisi L., Wang Y.-M., Johnson D. B. A Survey of Rollback-recovery Protocols in Message-passing Systems. *ACM Computing Surveys.* 2002. vol. 34, no. 3. pp. 375–408. DOI: 10.1145/568522.568525
8. Bouteiller A., Herault T., Bosilca G., Du P., Dongarra J. Algorithm-based Fault Tolerance for Dense Matrix Factorizations, Multiple Failures and Accuracy. *ACM Transactions on Parallel Computing.* 2015. vol. 1, no. 2. pp. 1–28. DOI: 10.1145/2686892
9. Engelmann C., Vallee G.R., Naughton T., Scott S.L. Proactive Fault Tolerance Using Preemptive Migration. *17th Euromicro International Conference on Parallel, Distributed and Network-based Processing (Weimar, Germany, February 18 – 20, 2009).* IEEE, 2009. pp. 252–257. DOI: 10.1109/PDP.2009.31.
10. Bondarenko A.A., Yakobovskiy M.V. Fault Tolerance for HPC by Using Local Checkpoints. *Vestnik Yuzho-Uralskogo gosudarstvennogo universiteta. Seriya Vychislitel'naya matematika i informatika* [Bulletin of South Ural State University. Series: Computational Mathematics and Software Engineering]. 2014. vol. 3, no. 3. pp. 20–36. DOI: 10.14529/cmse140302 (in Russian)
11. Di S., Bouguerra M.S., Bautista-Gomez L., Cappello F. Optimization of Multi-level Checkpoint Model for Large Scale HPC Applications. *28th International Parallel and Distributed Processing Symposium (Phoenix, Arizona, USA, May 19 – 23, 2014).* IEEE, 2014. pp. 1181–1190. DOI: 10.1109/IPDPS.2014.122.
12. Benoit A., Cavelan A., Le Fèvre V., Robert Y., Sun H. Towards Optimal Multi-level Checkpointing. *IEEE Transactions on Computers.* 2016. vol. 66, no. 7. pp. 1212–1226. DOI: 10.1109/TC.2016.2643660.
13. Di S., Robert Y., Vivien F., Cappello F. Toward an Optimal Online Checkpoint Solution under a Two-level HPC Checkpoint Model. *IEEE Transactions on Parallel and Distributed Systems.* 2016. vol. 28, no. 1. pp. 244–259. DOI: 10.1109/TPDS.2016.2546248.
14. *Fault Tolerance Research Hub.* Available at: http://fault-tolerance.org/ (accessed: 03.11.2018)
15. Bondarenko A.A., Lyakhov P.A., Yakobovskiy M.V. The Overheads Associated with Multi-level Coordinated Checkpointing. *Parallelnye vychislitelnye tekhnologii (PaVT'2017): Trudy mezhdunarodnoj nauchnoj konferentsii (Kazan', 3 – 7 aprelya 2017)* [Parallel Computational Technologies (PCT'2017): Proceedings of the International Scientific Conference (Kazan, Russia, 3 – 7 April, 2017)]. Chelyabinsk, Publishing of the South Ural State University, 2017. pp. 262–270. (in Russian)