Координированное сохранение контрольных точек с журналированием передаваемых данных и асинхронное восстановление расчетов после отказов*

А.А. Бондаренко[1], П.А. Ляхов[2], М.В. Якобовский[1]

ИПМ им. М.В.Келдыша РАН[1], МФТИ (ГУ) [2]

В работе описывается координированное сохранение контрольных точек с журналированием на уровне пользователя и последующее асинхронное восстановление в случае отказа в системе. Для этого метода определяются теоретические и экспериментальные величины накладных расходов на организацию отказоустойчивых вычислений. Проведено сравнение описанного метода с реализацией многоуровневого метода, основанного на координированном возврате всех процессов к последней контрольной точке. Предложенный метод на рассматриваемых наборах параметров, описывающих подверженность вычислительной системы отказам и характеризующих стратегии обеспечения отказоустойчивости, позволяет для алгоритмов с незначительным объемом данных для журналирования осуществить сокращение накладных расходов от 10% до 20 %.

Ключевые слова: параллельное программирование, MPI, расширение ULFM, контрольные точки, координированное сохранение, асинхронное восстановление, отказоустойчивые вычисления

1. Введение

В начале двухтысячных годов проходило масштабное наблюдение за отказами в вычислительных системах Лос-Аламоса. В нем было задействовано 22 кластера и около 5000 узлов. Результаты исследования показали, что за год в среднем в пересчете на один процессор приходится от 0.1 до 0.25 отказа в системе. 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, и даже если удастся увеличивать время на отказ для составляющих компонентов, в целом для суперкомпьютеров среднее время между отказами будет между 1 и 9 часами [4].

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 контрольных точек с журналированием передаваемых данных и асинхронном восстановлении расчетов после отказов. Предлагаемая стратегия сокращает время пересчета потерянных данных после отказа за счет использования дополнительных процессов.

В первой главе приведено описание многоуровневого метода обеспечения отказоустойчивости и предлагаемой модификации многоуровневого метода. Во второй главе приводятся особенности реализации предлагаемой модификации. В третьей главе содержатся оценки наклад-

* Работа выполнена при поддержке РФФИ по гранту 17-07-01604 а.

ных расходов, полученные теоретическим образом и в результате вычислительных экспериментов.

2. Многоуровневый метод обеспечения отказоустойчивости

2.1. Предпосылки использования многоуровневого метода обеспечения отказоустойчивости

Следующие аргументы послужили причиной разработки многоуровневого метода обеспечения отказоустойчивости.

- В системе могут происходить программные сбои, отказы в вычислительных составляющих или в коммуникационном оборудовании [5], которые могут иметь разные частоты отказов, разное время восстановления так и разные число недоступных для коммуникации mpi-процессов после отказа.
- Хорошо изучены различные протоколы сохранения контрольных точек и последующего восстановления [6,7]. Существуют алгоритмические методы обеспечения отказоустойчивости (algorithm-based fault tolerance) [8]. Разрабатываются методы дублирования расчетов, а также предсказания наступления отказов по состоянию системы [9] и последующего исключения потенциально опасного компонента системы. Объединение и комбинирование этих методов могут дополнительно уменьшить накладные расходы при обработке отказов.
- Сохранение контрольных точек может осуществляться в разные средства хранения: оперативную память, локальные диски (HDD, SSD), распределенную файловую систему. Применение различных стратегий дублирования данных к различным локальным устройствам хранения [10] могут дополнительно уменьшить накладные расходы на этапе сохранения данных.

Наличие этих аргументов привело к разработке многоуровневого подхода, использующего разные алгоритмы обеспечения отказоустойчивости, разнообразные стратегии сохранения контрольных точек и последующего восстановления, которые позволяют 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 от 1 до k уровней (рассматриваемых типов) отказа;
- наступление отказов на разных уровнях – независимые случайные величины;
- каждый уровень j имеет свою частоту отказа λ_j, свои накладные расходы на сохранение контрольной точки $\mathrm{C_j}$ и восстановление после отказа R_j;
- отказ на уровне j уничтожает все контрольные точки на нижних уровнях от 1 до $j-1$ и приводит к необходимости осуществлять восстановление с уровня j или выше;
- восстановление на уровне j восстанавливает данные контрольных точек для всех нижних уровней;
- частота отказов убывает, а накладные расходы на сохранение и восстановление возрастают с ростом уровня, то есть $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_\mathrm{k}$, $\mathrm{C}_1 \leq \mathrm{C}_2 \leq \cdots \leq \mathrm{C}_\mathrm{k}$ и $\mathrm{R}_1 \leq \mathrm{R}_2 \leq \cdots \leq \mathrm{R}_\mathrm{k}$.

На рисунке 1 представлена общая процедура запуска приложения с тремя случаями отказа, где T (базовое время) – время выполнения исходной программы реализованной без средств обеспечения отказоустойчивости, x_i число интервалов между сохранениями контрольных точек на i-ом уровне в течение T. Для упрощения представления на рисунке отсутствуют: период восстановление системы, накладные расходы на сохранение контрольных точек и восстановление из соответствующей точки в случае отказа.

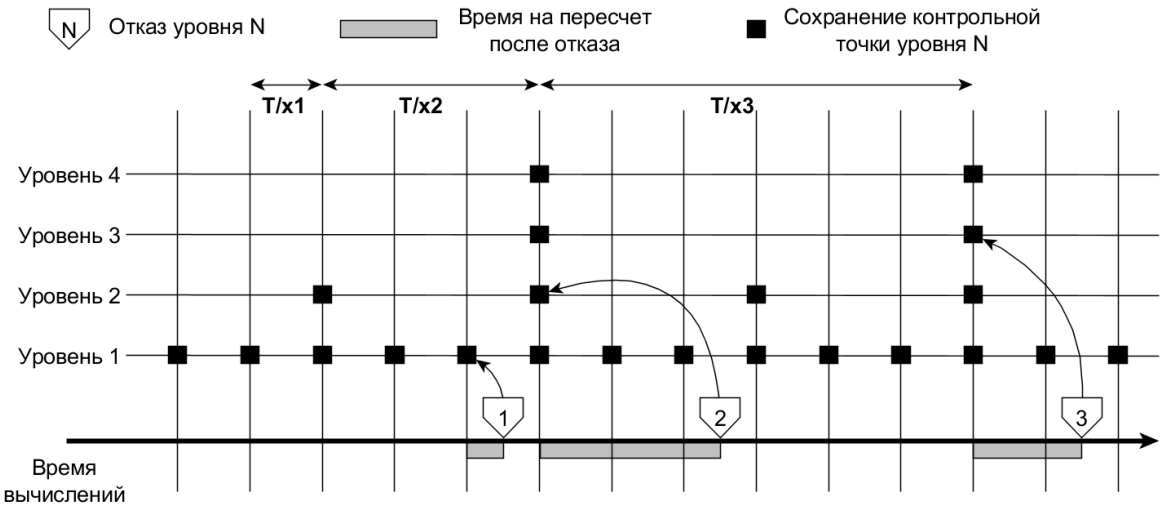


Рис. 1. Исполнение HPC-приложения с отказами

Замечание 1. Ниже будем использовать следующие выражения: отказ *j-го* уровня, сохранение *j-го* уровня, восстановление *j-го* уровня. Под данными выражениями будем понимать, что для многоуровневого метода выбран набор отказов для каждого уровня и были выбраны стратегии сохранения и восстановления для соответствующего отказа, удовлетворяющие 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].

В качестве примера многоуровневых методов обеспечения отказоустойчивости приведем четырехуровневый метод, в котором авторы работы [12] имеют дело со следующими отказами:

- на первом уровне – временные ошибки в памяти;
- на втором уровне – сбои в узлах, которые делают недоступными данные из оперативной памяти;
- на третьем уровне – сбои, приводящие к недоступности данных из оперативной памяти и данные из резервных копий соседних узлов;
- на четвертом уровне – сбои, которые делают недоступными данные, сохраненные вышеприведенными методами.

Сохранение ключевых данных для первого уровня будет осуществляться в оперативную память, для второго уровня осуществляется дублирование данных некоторым соседям, для третьего уровня осуществляется локальное сохранение данных с применением кодов Рида — Соломона, для четвертого уровня сохранение осуществляется в распределенную файловую систему.

2.2. Сохранение и восстановление для многоуровневого метода

Организация сохранения контрольных точек оптимальным образом для всего времени счета является нетривиальной задачей и выходит за рамки данной работы. Различные подходы к решению данного вопроса представлены в работах [11-13]. Принято выделять некий шаблон сохранения, в рамках которого осуществляется оптимизация, а сохранение контрольных точек на протяжении всего счета осуществляется за счет периодического повторения найденного оптимального шаблона. На рис. 2 приведена иллюстрация организации координированного сохранения контрольных точек для двухуровневого метода обеспечения отказоустойчивости с произвольным шаблоном (оптимизация построения шаблонов для 2 уровней изложена в работе

[13]). Для больших уровней обеспечения отказоустойчивости подход к оптимизации шаблонов представлен в работе [12].

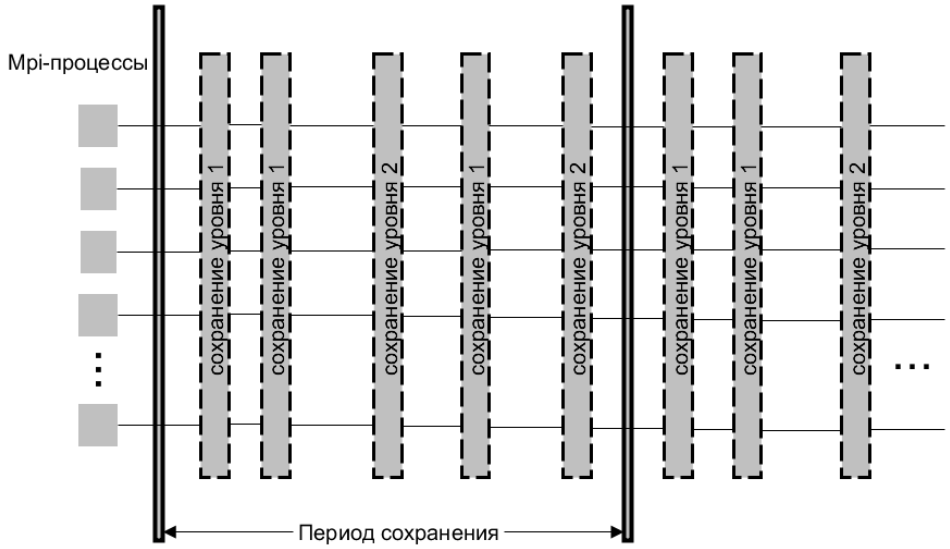


Рис. 2. Координированное сохранение контрольных точек для двухуровневого метода обеспечения отказоустойчивости

Для приведенных этапов сохранения осуществляется одновременная запись контрольных точек в соответствующее устройство памяти или специальным образом соответствующим выбранному уровню (например, с применением кодов Рида-Соломона). Выбранные с самого начала периоды сохранения повторяются до конца счета или пока не произойдет отказ, после восстановления от которого периоды начинают повторяться снова.

Процедура восстановления после отказа (см. рис. 4. а) включает в себя следующие этапы:

Шаг 1. Обнаружение наличия отказа в системе, распространение информации об отказе, определение количества отказавших процессов и их номеров.

Шаг 2. Перенастройка MPI среды для нормальной работы коммуникационных функций.

Шаг 3. Взамен отказавшим mpi-процессам для проведения расчетов вводятся резервные mpi-процессы (или новые процессы, если возможна генерация mpi-процессов), таким образом, что новые процессы имеют номера отказавших процессов.

Шаг 4. Теперь для возобновления работы все mpi-процессы должны осуществить чтение данных из последней контрольной точки.

Шаг 5. Осуществляется запуск продолжения расчетов.

В соответствии со статьей [11] полное время работы программы реализующей многоуровневый метод обеспечения отказоустойчивости можно оценить формулой:

$$T_{\text{итоговое}} = T_{\text{б}} + \sum_{i=1}^{k} C_i(x_i - 1) + \sum_{i=1}^{k} \left(\frac{T_e}{2x_i} + \sum_{j=1}^{i-1} \frac{C_j x_j}{2x_j} + D_i + R_i \right) n_i$$

Первое слагаемое в формуле – базовое время, время работы программы без средств обеспечения отказоустойчивости. Второе слагаемое в формуле – сумма накладных расходов по сохранению контрольных точек для всех уровней. Третье слагаемое – сумма накладных расходов на восстановление после отказов соответствующего уровня, где первое слагаемое – ожидаемый период пересчета после отказа; второе слагаемое – сумма накладных расходов на восстановление контрольных точек нижних уровней; третье и четвертое слагаемые – накладные расходы на аппаратное восстановление, восстановление MPI и восстановление данных из контрольной точки.

Для записи формулы и далее используются обозначения:

- $T_{итоговое}$ – суммарное время работы программы с учетом наличия отказов в системе;

- $T_б$ – время на полезные вычисления в ходе программы или время работы программы без средств обеспечения отказоустойчивости;
- C_i – накладные расходы на сохранение контрольной точки на i-ом уровне;
- x_i – число периодов времени между контрольными точками на i-ом уровне;
- n_i – ожидаемое число отказов на i-ом уровне за время работы программы;
- D_i – восстановление системы после отказа i-ого уровня (определение функционирующих элементов системы, восстановление параллельной среды выполнения и др.),
- R_i – время на восстановление данных из контрольной точки i-го уровня.

Замечание 3. Так как отказы являются случайными величинами и сохранение контрольных точек происходит периодически через одинаковые промежутки времени, то для отказа уровня j в качестве оценки ожидаемого времени потерянных вычислений (времени пересчета) после отказа примем половину интервала между контрольными точками $T_б/(2x_j)$.

2.3 Модификация многоуровневого метода

Ключевой особенностью предлагаемого метода является сохранение информации о передаваемых данных между процессами. На всем протяжении времени, пока не будет вновь осуществлена запись новых контрольных точек, необходимо хранить журнал с переданными данными. Таким образом, в случае отказа все живые процессы могут находиться в ожидании, так как они владеют информацией о последней итерации расчетов, а сохраненные передаваемые данные со времени последнего сохранения контрольной точки помогут помочь осуществить асинхронное восстановление процессам, занявшим место отказавших. Для некоторых расчетов можно осуществить запуск вместо одного отказавшего процесса нескольких, что позволит ускорить процесс восстановления потерянных расчетов.

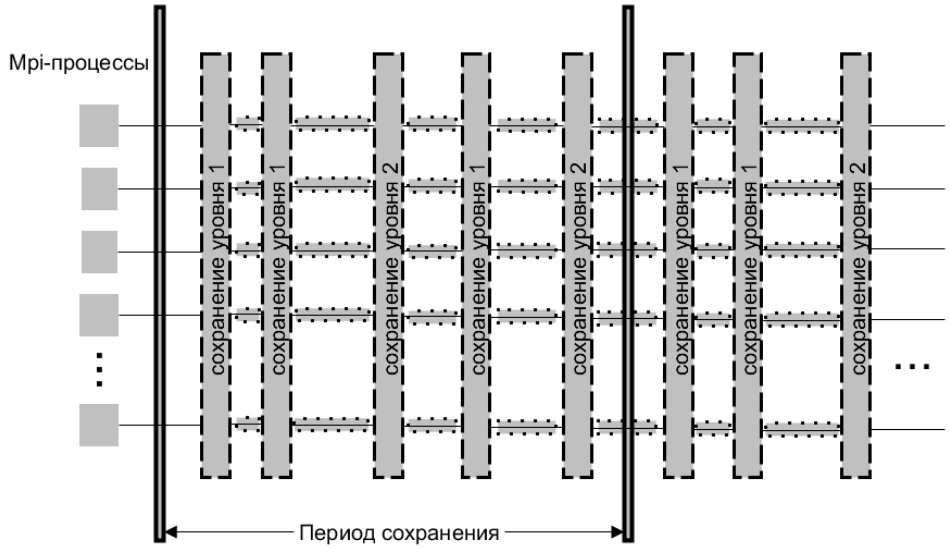


Рис. 3. Сохранение контрольных точек с журналированием передаваемых данных для двухуровневого метода обеспечения отказоустойчивости

Рис. 3 иллюстрирует организацию координированного сохранения контрольных точек с журналированием (отмечено серой линией с точками) передаваемых данных для двухуровневого метода обеспечения отказоустойчивости. Для приведенных этапов сохранения осуществляется одновременная запись контрольных точек в соответствующее устройство памяти или специальным образом, соответствующим выбранному уровню. Между координированным сохранением контрольных точек должно осуществляться сохранение информации о передаваемых данных между процессами. Будем полагать объем передаваемых данных соседям существенно меньшим объема данных для работы одного mpi-процесса. Как следствие можно заложить в

программу, чтобы журналирование осуществлялось в оперативную память, или в другие доступные локальные устройства хранения.

Общая схема процедуры асинхронного восстановления (см. рис.4. б) после отказа включает в себя следующие шаги.

Шаг 1. Обнаружение наличия отказа в системе, распространение информации об отказе, определение количества отказавших процессов и их номеров.

Шаг 2. Перенастройка MPI среды для нормальной работы коммуникационных функций.

Шаг 3. Взамен отказавшим mpi-процессам для проведения расчетов вводятся резервные mpi-процессы (или новые процессы, если возможна генерация mpi-процессов), таким образом, что новые mpi-процессы имеют номера отказавших mpi-процессов.

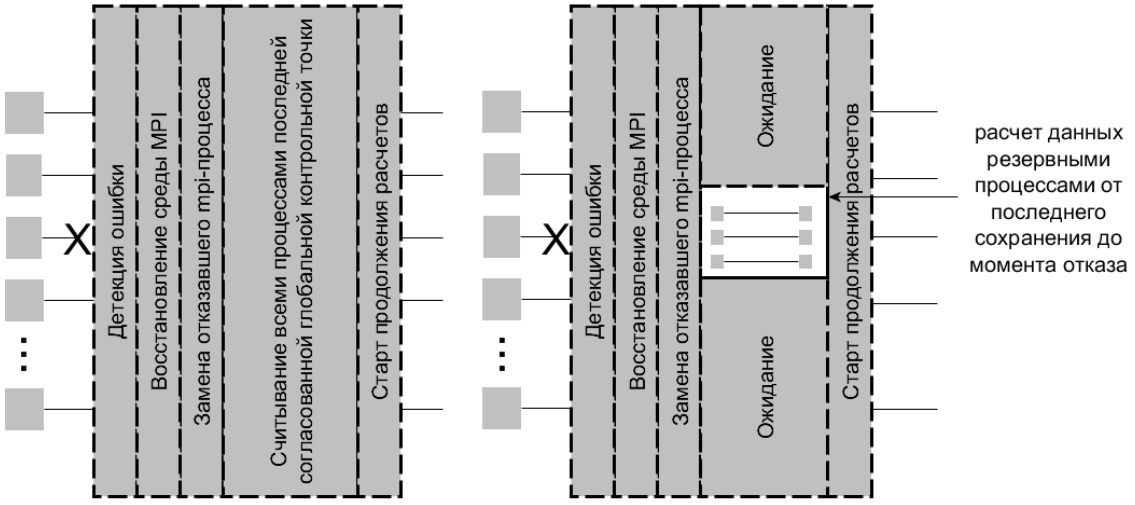
Шаг 4. Не отказавшие процессы находятся в ожидании, за исключением некоторых, которые должны передать журнальные данные с момента последнего сохранения контрольной точки. Группа резервных процессов осуществляет пересчет потерянных расчетов с последней контрольной точки до момента отказа. На пересчет выделяются резервные mpi-процессы в кратном размере $w \in \{2, 5, 10\}$ на один отказавший процесс, чтобы ускорить процесс восстановления по сравнению с восстановлением за счет одновременного отката всех mpi-процессов к последней контрольной точке. Для осуществления пересчета резервные процессы будут получать передаваемые данные от «некоторых» живых mpi-процессов от момента последней сохраненной контрольной точки до отказа.

Шаг 5. Осуществляется переход управления не отказавшим mpi-процессам и пришедшим на замену отказавшим, с последующим запуском продолжения расчетов.

Общее время работы предлагаемой модификации отличается от стандартной реализации сокращением времени на пересчет при восстановлении после отказа и представляется следующей формулой:

$$T_{\text{итоговое}} = T_{\text{б}} + \sum_{i=1}^{k} C_i(x_i - 1) + \sum_{i=1}^{k} \left(\frac{T_e}{2x_i} / w_i + \sum_{j=1}^{i-1} \frac{C_j x_j}{2x_j} + D_i + R_i \right) n_i \,,$$

где w_i – количество резервных процессов принимающих участие в пересчете работы одного отказавшего процесса, i – уровень отказа.



а) восстановление стандартной реализации б) восстановление предлагаемой модификации

Рис. 4. Схема восстановлений многоуровневого метода и предлагаемой модификации

Замечание 4. Для возможности выполнения алгоритма необходимо, чтобы между периодами сохранения контрольных точек каждый процесс хранил все передаваемые им данные. Пусть $V_{\text{жур}}$ – объем передаваемых данных одним процессом за вычислительную итерацию, L – количество вычислительных итераций между сохранениями контрольных точек. Если произве-

дение $V_{жур} \cdot L$ 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.

3. Особенности реализации алгоритма

3.1 Получение параметров для многоуровневого метода

Для реализации многоуровневого метода необходимо определить ключевые параметры метода. Выбираем отказы и соответствующие алгоритмы сохранения и восстановления. Предполагаем отказы независимыми. Пусть известно среднее время между отказами (MTBF). Для выбранной вычислительной задачи определяем набор ключевых данных, необходимых для осуществления восстановления счета одного или нескольких процессов в случае отказа. После этого определяем характеристики записи контрольных точек для каждого уровня C_i, а также значения скорости чтения этих данных с соответствующего носителя R_i.

Далее необходимо определить шаблон сохранения ($W^{опт}$, $N_i^{опт}$) этих данных, для этого мы используем теорему 2 из работы [12]. Данная теорема позволяет получить алгоритм (см. рис. 5) для определения оптимальной длинны шаблона и частот сохранения контрольных точек разных уровней. После определения этих параметров можно приступать к вычислениям.

```
void optimal_pattern(double *lambda, double *N_opt, double *C, int
count, double &W_opt){
      double S_bop(0), S_top(0);
      for(int i=0;i<count;i++){
            N_opt [i] = sqrt(C(count-1)*lambda(i)/lambda(count-1)/C(i));
            S_top += N_opt (i)*C(i);
            S_bot += lamda(i)/ N_opt (i);
      }
      W_opt = sqrt(2*S_top/S_bot);
}
```

Рис. 5. Алгоритм определения оптимальных параметров шаблона сохранения

3.2 Моделирование отказов и восстановление среды MPI

Отказы рассматриваются как случайные величины, описываемые экспоненциальным распределением. Для реализации события наступления отказа используем алгоритм (см. рис. 6), осуществляющий генерацию списка случайных величин, следующих друг за другом и подчиняющихся экспоненциальному распределению.

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

Рис. 6. Алгоритм 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 осуществления обнаружения наличия отказа в системе, распростране-

ния информации об ошибке и последующем восстановлении среды MPI будем использовать функционал ULFM-1.1 Release – ответвление open-MPI с расширением ULFM [14]. Примеры реализации программ с ULFM и описанием его функционала можно найти в [14, 15].

3.3 Краевая задача распределения тепла в тонкой прямоугольной пластине

В данной работе для вычислительного эксперимента выбирается краевая задача распределения тепла в тонкой прямоугольной пластине. Используем явную схему для проведения вычислений 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.

Отметим, что в данной работе будем рассматривать асинхронное восстановление только на первом уровне с числом резервных процессов, равным 2 и 5. Реализации асинхронного восстановления на более высоких уровнях отказа ограничивается объемом данных для журналирования каждым процессом и требует дополнительного исследования для вычислительных алгоритмов и характеристик выбранных вычислительных систем.

Выберем следующие отказы: на первом уровне – незначительные сбои в системе, для которых достаточно данных, хранящихся в оперативной памяти соседнего процесса, на втором уровне – более сложные сбои, для восстановления после которых необходимо хранить данные в распределенной файловой системе. Сохранение контрольных точек осуществляем для первого уровня в оперативную память соседнего процесса и для второго уровня в распределенную файловую систему. Каждый процесс записывает к себе в оперативную память данные, которые он передает соседним процессам. В случае отказа первого уровня не отказавшие процессы ожидают, а пересчет осуществляют 2 или 5 резервных процесса, между которыми равномерно делятся данные для пересчета. В случае отказа второго уровня восстанавливается MPI среда, к расчету подключается новый процесс, взамен отказавшего, все процессы считывают последнюю доступную глобальную контрольную точку. Расчет продолжается до тех пор, пока не будет произведен весь базовый расчет, то есть не пройдет все время базового счета.

4. Теоретическая и экспериментальная оценки накладных расходов

Рассмотрим группу тестов для двухуровневых методов обеспечения отказов и проведем теоретические оценки сокращения накладных расходов при использовании {2, 5, 10} резервных процессов. Данная группа тестов подробно рассматривается в работе [13]. Теоретическая оценка (таблица 1) показывает, что сокращение времени пересчета при отказе на первом уровне позволяет уменьшить суммарные накладные расходы от 10% до 20%.

В таблицах 1 и 2 используются дополнительные к приведенным в параграфе 2.2 условные обозначения: $T_{нр}$ – оценка накладных расходов, то есть разница между оценкой времени выполнения программы с отказами и временем базовых вычислений, ΔT – оценка сокращенного времени за счет использования предлагаемой модификации многоуровневого метода, % – отношение ΔT к $T_{нр}$, w_1 – число резервных процессов, принимающих участие в пересчете потерянной работы одним процессом из-за отказа.

Для вычислительного эксперимента выберем параметры расчета таким образом, чтобы число отказов за время базового расчета было равно 20, 50, 100, а также, чтобы на каждые 9 отказов первого уровня ожидался 1 отказ второго уровня. Время базового расчета было выбрано равным 1 часу, а соответствующие длины шаблонов получились следующими $W^{опт}$ = 135 с, 85 с и 60 с, а сам шаблон – {10, 1}, то есть на каждые 10 итераций сохранения контрольных точек первого уровня приходится 1 сохранение контрольной точки второго уровня.

Таблица 1. Теоретическая оценка сокращения накладных расходов при 2 уровнях отказов

Количество резервных процессов	w_1 = 2	w_1 = 5	w_1= 10

Параметры теста (единица времени – секунда)	$T_{нр}$	ΔT	%	ΔT	%	ΔT	%
#1 $T_б$= 86400, C_1 = 20, C_2 = 50, x_1= 224, x_2= 60, n_1= 24, n_2= 4	15748	2314	15	3703	24	4166	27
#2 $T_б$ = 86400, C_1 = 20, C_2 = 50, x_1 =320, x_2 =96, n_1 =50, n_2 =10	24213	3375	14	5400	22	6075	25
#3 $T_б$ = 86400, C_1 = 20, C_2 = 100, x_1 =442, x_2 =97, n_1 =100, n_2=20	42012	4887	12	7819	20	8796	21
#4 $T_б$ = 86400, C_1 = 10, C_2 = 40, x_1=636, x_2 =152, n_1=100, n_2=20	27085	3396	13	5434	20	6113	23
#5 $T_б$ = 86400, C_1 = 20, C_2 = 40, x_1=889, x_2=218, n_1=200, n_2=40	39621	4859	12	7775	20	8747	23
#6 $T_б$ = 43200, C_1 = 10, C_2 = 40, x_1=434, x_2 =68, n_1 =100, n_2=20	25998	2489	10	3982	15	4479	17
#7 $T_б$ = 21600, C_1 = 40, C_2 = 200, x_1 =127, x_2 = 31, n_1 = 75, n_2= 15	29873	3189	11	5102	17	5740	19
#8 $T_б$ = 21600, C_1 = 50, C_2 =300, x_1 =129, x_2 = 26, n_1 =100, n_2= 15	39863	4186	11	6698	17	7535	19
#9 $T_б$ = 10800, C_1 = 50, C_2 = 300, x_1 = 64, x_2 = 13, n_1 = 50, n_2= 7	19338	2109	11	3375	17	3797	20

Таблица 2. Теоретическая и экспериментальная оценки расчета тестового примера

Параметры задачи (единица времени – секунда)	Теоретическая оценка		Вычислительный эксперимент		Теоретическая оценка		Вычислительный эксперимент	
	w_1 =2				w_1=5			
#1 $T_б$ = 3600 C_1 = 0.5, C_2=5, R_1 = 0.3, R_2=3, MTBF ={200, 1800} $W^{опт}$ = 135, Шаблон {10, 1}	$T_{нр}$		$T_{нр}$		$T_{нр}$		$T_{нр}$	
	473		851		437		871	
	ΔT	%	ΔT	%	ΔT	%	ΔT	%
	61	11	65	8	97	18	131	15
#2 $T_б$ = 3600 C_1 = 0.5, C_2=5, R_1 = 0.3, R_2=3, MTBF ={80, 720} $W_{опт}$ = 85, Шаблон {10,1}	$T_{нр}$		$T_{нр}$		$T_{нр}$		$T_{нр}$	
	767		1038		710		958	
	ΔT	%	ΔT	%	ΔT	%	ΔT	%
	96	12	109	11	153	18	181	19
#3 $T_б$ = 3600 C_1 = 0.5, C_2=5, R_1 = 0.3, R_2=3, MTBF ={40, 360} $W_{опт}$ = 60 Шаблон {10,1}	$T_{нр}$		$T_{нр}$		$T_{нр}$		$T_{нр}$	
	1112		1663		1031		1438	
	ΔT	%	ΔT	%	ΔT	%	ΔT	%
	135	11	145	9	216	17	248	17

В таблице 2 приведены результаты теоретической оценки и вычислительного эксперимента для рассматриваемых параметров. При увеличении частоты отказов (уменьшении MTBF) объем накладных расходов растёт. При накладных расходах на сохранение одной контрольной точки менее 0,15% и на восстановление менее 0,1% от базового времени счета, объём накладных расходов колебался между 25% (для ожидаемого суммарного числа отказов равного 20) и

50% (для ожидаемого суммарного числа отказов равного 100). Вычислительные эксперименты показали, что применение 2 резервных процессов для пересчета потерянных данных одного процесса после отказа первого уровня, позволяет уменьшить объем всех накладных расходов более чем на 5%, а применение 5 резервных процессов позволяет уменьшить объем всех накладных расходов на 10% – 20%.

Отметим наличие существенной разницы в оценке накладных расходов между теоретической оценкой и вычислительным экспериментом. Данная разница связана с тем, что для теоретической оценки количество отказов фиксировано и определяется объемом базовых вычислений, то есть не рассматривается возможность отказа в возникающих накладных расходах. А в вычислительном эксперименте измеряются 2 параметра времени: $T_{бв}$ – время расчета базовых вычислений и $T_{тв}$ – текущее время работы всей программы, включая накладные расходы на сохранение и восстановление после отказа. Параллельная программа выполняется пока не будет выполнен базовый расчет $T_{бв} \geq T_{б}$, а наступление событий сохранения контрольных точек и отказа разных уровней фиксируется по текущему времени $T_{тв}$. Как следствие для теоретического расчета число отказов было 20, 50 и 100, а для соответствующих параметров в вычислительном эксперименте число отказов было – 25, 64, 161.

Заключение

Результаты теоретической оценки и вычислительного эксперимента для рассматриваемых параметров показывают, что применение 5 резервных процессов для пересчета потерянных данных одного процесса после отказа первого уровня, позволяет уменьшить объем всех накладных расходов от 10% до 20%. Данный результат позволяет говорить о целесообразности применения предлагаемой модификации для алгоритмов с незначительным объемом данных для журналирования, однако возможность ее применения для других алгоритмов требует дальнейших исследований.

Литература

1. Gibson G. Failure in Supercomputers and Supercomputer Storage Los Alamos Computer Science Symposium, Oct 15, 2008 URL: http://www.csm.ornl.gov/srt/conferences/ResilienceSummit/2008/pdf/gibson_slides.pdf (дата обращения: 15.04.2018)
2. Reed D. High-end computing: The challenge of scale. Director’s Colloquium, Los Alamos National Laboratory, May 2004.
3. Di Martino, C., 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 // In Dependable Systems and Networks 44th Annual IEEE/IFIP International Conference on, DSN 2014, June 23–26, 2014, Atlanta, Georgia, USA. IEEE, 2010 P. 610–621. DOI 10.1109/DSN.2014.62
4. Dongarra J., Herault T., Robert Y. Fault-Tolerance Techniques for High-Performance Computing. Springer, 2015. 320 p. DOI: 10.1007/978-3-319-20943-2
5. Sorin D. Fault Tolerant Computer Architecture. Synthesis Lectures on Computer Architecture Morgan&Claypool, 2009. 104 p. DOI: 10.2200/S00192ED1V01Y200904CAC005
6. 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
7. 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
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. 28. DOI: 10.1145/2686892

9. Engelmann, C., Vallee, G.R., Naughton, T., Scott, S.L. Proactive fault tolerance using preemptive migration / In Proceedings of the 17th Euromicro International Conference on Parallel, Distributed, and network-based Processing, PDP 2009, February 18-20, 2009, Weimar, Germany. 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., No. 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 //Proceedings of the 28th International Parallel and Distributed Processing Symposium, IPDPS 2014, May 19-23, 2014, Phoenix, Arizona, USA. 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. Preprint. IEEE Trans. Parallel and Distributed Systems. 2016. URL: https://hal.inria.fr/hal-01263879/document. (дата обращения: 15.04.2018). DOI: 10.1109/TPDS.2016.2546248.

14. Fault Tolerance Research Hub [Электронный ресурс] Режим доступа: http://fault-tolerance.org/ (дата обращения: 15.04.2017)

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 – XI международная конференция, ПаВТ'2017, г. Казань, 3–7 апреля 2017 г. Короткие статьи и описания плакатов. Челябинск: Издательский центр ЮУрГУ, 2017. С. 262–270.