Восстановление расчетов методом диффузной балансировки нагрузки*

А.А. Бондаренко, С.К. Григорьев, М.В. Якобовский

ИПМ им. М.В. Келдыша РАН

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, что приводит к необходимости разработки эффективных подходов и алгоритмов обеспечения отказоустойчивых расчетов. В данной работе рассматривается проблема возобновления расчетов на меньшем числе процессов, после отказа в системе. В работе рассматривается два решения. Первое заключается в том, что в начале расчета выделяются запасные процессы, которые используются для замены отказавших процессов. Второе решение основывается на применении диффузной балансировки для перераспределения решаемой задачи на меньшее число процессов. Для удобства проведения сравнения этих методов в экспериментах отказы происходят через заданные интервалы времени. Теоретические и экспериментальные оценки накладных расходов демонстрируют, что при небольшом числе отказов в течение расчета для восстановления выгоднее использовать стратегию с балансировкой нагрузки.

Ключевые слова: диффузная балансировка нагрузки, расширение ULFM, контрольные точки, отказоустойчивость.

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 [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.

При отказе в системе после восстановления MPI среды возникает необходимость решения задачи возобновления расчета на меньшем числе процессов. Естественным решением данной задачи является выделение запасных процессов, которые должны заменить отказавшие процессы. В работах [5–8] рассмотрены методы сохранения контрольных точек и восстановления расчетов

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

после отказов при выделении группы запасных процессов. В данной работе рассматривается применение диффузной балансировки для перераспределения решаемой задачи на меньшее число процессов. Цель данной работы заключается в сравнении этих методов восстановления и определения условий, при которых применение балансировки позволяет сократить накладные расходы на обеспечение отказоустойчивости расчетов.

Во втором разделе изложен многоуровневый метод обеспечения отказоустойчивости и теоретическая оценка накладных расходов при фиксированном числе отказов. В третьем разделе описано применение метода диффузной балансировки нагрузки для организации восстановления после отказов в системе. В четвертом разделе приводятся результаты вычислительных экспериментов, а именно оценки накладных расходов для разных стратегий восстановления.

2. Отказоустойчивость параллельных программ

Отказы в вычислительных системах могут приводить к различным последствиям для исполняемых параллельных программ. В этом разделе рассматриваются инструментарий прикладного программиста, который дает возможности реагировать на эти последствия.

2.1 Отказы в вычислительных системах

Современные вычислительные системы состоят из сложных компонентов, которые объединены в блоки и имеют сложные средства передачи сообщений. Некорректное функционирование системного или прикладного ПО, дефекты изготовления или дизайна аппаратных составляющих, физическое повреждения или изнашивание аппаратных составляющих, воздействие космического излучения, воздействие альфа частиц, внешнее электромагнитное воздействие – все эти причины могут приводить к отказам в вычислительных системах. Однако разработчики аппаратных составляющих будут рассматривать отказы под своим углом зрения (на своем уровне абстракции), а разработчики системного ПО под своим углом.

Обеспечение отказоустойчивости на системном уровне, основанное на BLCR, заключается в записи всей памяти приложения в глобальную контрольную точку и последующем перезапуске программы из этой контрольной точки в случае отказа. Операции сохранения и перезапуска могут быть выполнены автоматически, поскольку не зависят от специфики прикладной программы. Преимуществом автоматических операций сохранения и перезапуска является простота использования, так как от пользователя не требуется внесения изменений в код программы. Однако эта простота достигается за счет очевидного недостатка — высоких накладных расходов, так как в контрольную точку записывается все данные, используемые приложением, в том числе данные, которые не являются существенными для запуска программы из контрольной точки. В результате нескольких независимых исследований [1, 9, 10] было предсказано, что потенциальные Экзафлопсные вычислительные системы будут тратить более 50% своего времени на чтение или запись контрольных точек. Такие опасения побудили исследователей к разработкам новых методов, в том числе повышающих масштабируемость техник сохранения контрольных точек.

Один из подходов, уменьшающих накладные расходы работы с контрольными точками, состоит в том, чтобы предоставить пользователю инструментарий по обеспечению отказоустойчивости. В этом случае в контрольную точку будет входить только то, что явно укажет прикладной программист. В идеале, только те данные, которые необходимы для восстановления утерянной в результате сбоя информации. Так же на уровне пользователя контролируется частота и момент создания контрольных точек при выполнении программы. То есть накладные расходы могут быть значительно уменьшены, однако, потребуется дополнительная работа прикладного программиста для реализации отказоустойчивости в приложении.

При рассмотрении отказов «аварийная остановка» (fail-stop failures) группа разработчиков ULFM (FT-MPI) [11, 12] предлагает прикладному программисту под ними понимать невозможность осуществления коммуникационных операций с одним или несколькими mpi-процессами. Однако для этого программные средства обмена сообщениями (MPI) должны обеспечивать возможность коммуникации между вычислительными процессами даже при наличии отказов в вы-

числительной системе, а также обеспечивать способность восстановления приложения в согласованное состояние, из которого вычисления могут быть продолжены. Стандартная версия MPI (MPI 3.1) не содержит средств обработки отказов при коммуникационных операциях.

На данный момент ULFM – расширение стандарта MPI, позволяющее реализовывать методы обеспечения отказоустойчивости на уровне пользователя, находится в разработке. Описание расширения ULFM и его функционала можно найти в [11], например, для обнаружения отказа в системе, распространения информации об ошибке и последующем восстановлении среды MPI предлагаются использовать функции: MPIX_Comm_agree(), MPIX_Comm_revoke(), MPIX_Comm_shrink(). Программные реализации предварительных версий стандарта MPI с расширением ULFM были представлены на конференциях SC'14-SC'17.

2.2 Двухуровневый метод 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 [12], 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 [13, 14]. Существуют алгоритмические методы обеспечения отказоустойчивости (algorithm-based fault tolerance) [15]. Разрабатываются методы дублирования расчетов, а также предсказания наступления отказов по состоянию системы [16] 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 [5] могут дополнительно уменьшить накладные расходы на этапе сохранения данных.

Наличие этих аргументов привело к разработке многоуровневых методов обеспечения отказоустойчивости, основанных на разных алгоритмах обеспечения отказоустойчивости и разнообразных стратегиях сохранения контрольных точек, 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.

В данной работе 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 необходимо хранить данные в распределенной файловой системе. Сохранение ключевых данных для первого уровня будет осуществляться в оперативную память с дублирование одному соседу, а для второго уровня сохранение будет осуществляться в распределенную файловую систему.

Шаг 1. Происходит оповещение всей системы и программы о наличии отказа (с помощью ULFM функции – MPI_COMM_REVOKE).
Шаг 2. Осуществляется восстановление MPI (с помощью ULFM функции – MPI_COMM_SHRINK).
Шаг 3. Осуществляется замена отказавших mpi-процессов на запасные, если в начале работы программы часть процессов была выведена из расчетов и помечена как запасные.
Шаг 4. Для восстановления расчетов осуществляется возврат всех mpi-процессов к последней контрольной точке соответствующего уровня.

Рис. 1. Алгоритм восстановления расчетов, основанный на диффузной балансировке

Организация сохранения контрольных точек оптимальным образом для всего времени счета, является не тривиальной задачей и выходит за рамки данной работы. Различные подходы к решению данного вопроса представлены в работах [17, 18]. 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. Определение параметров шаблона сохранения происходит по теореме 2 работы [178].

Для обнаружения отказа после каждой локальной или глобальной операции обмена данными осуществляется проверка (с помощью ULFM функции – MPI_COMM_AGREE) на наличие отказа в системе, если его нет, то продолжается выполнение основного алгоритм программы. Если обнаружен отказ, то осуществляется переход к алгоритму восстановления см. рис. 1.

2.3 Теоретическая оценка накладных расходов при фиксированном числе отказов

В соответствии со статьей [19] полное время работы программы реализующей двухуровневый метод обеспечения отказоустойчивости можно оценить формулой:

$$T_{\text{итоговое}} = T_{\text{б}} + C_1(x_1 - 1) + C_2(x_2 - 1) + \left(\frac{T_{\text{б}}}{2x_1} + D + R_1\right)n_1 + \left(\frac{T_{\text{б}} + C_1x_1}{2x_2} + D + R_2\right)n_2 \qquad (1)$$

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 отказов в системе;
- $T_{\text{б}}$ — 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 — восстановление системы (определение функционирующих элементов системы, восстановление параллельной среды выполнения и др.),
- R_i — время на восстановление данных из контрольной точки i-го уровня.

Так как отказы являются случайными величинами, а сохранение контрольных точек происходит периодически через одинаковые промежутки времени, то для отказа j-го уровня в качестве оценки ожидаемого времени потерянных вычислений (времени пересчета) после отказа принята половина интервала между контрольными точками $T_{\text{б}}/2x_j$.

Замечание 1. В вычислительном эксперименте отказы происходят в заранее выбранные моменты времени, однако в теоретической оценке накладных расходов время повторного пересчета принято равным половине интервала между сохранениями контрольных точек.

3. Диффузная балансировка нагрузки

3.1 Восстановление расчетов методом диффузной балансировки

В рамках метода диффузной балансировки нагрузки [20] перераспределение вычислительной нагрузки выполняется преимущественно между логически соседними процессорами. Метод эффективен при решении задач с медленно меняющимся распределением вычислительной нагрузки и позволяет сохранить важное свойство «локальности» вычислительного алгоритма, не требуя непосредственного взаимодействия каждого из процессоров со всеми остальными.

Пусть V — объем всей сетки, P — число отказавших процессов, тогда до отказа каждый процесс работал и сохранял в контрольные точки с объем V/(N), а после отказа процессы должны работать с объемом V/(N-P). Таким образом, задача балансировки вычислительной нагрузки сводится к задаче о сборе данных из контрольных точек.

До отказа каждый процесс с номером i проводил вычисление с частью сетки (a_i, a_{i+1}), а после восстановления процессы получают новые номера j и проводят вычисления с большим объемом сетки (b_j, b_{j+1}). Каждый процесс определяется парой (i, j). Отметим, что данные, с которыми работал отказавший процесс, для отказов первого уровня доступны в оперативной памяти соседнего процесса, а для отказов второго уровня – в распределенной файловой системе. На рис. 2 представлена схема распределения данных до отказа и после балансировки нагрузки.

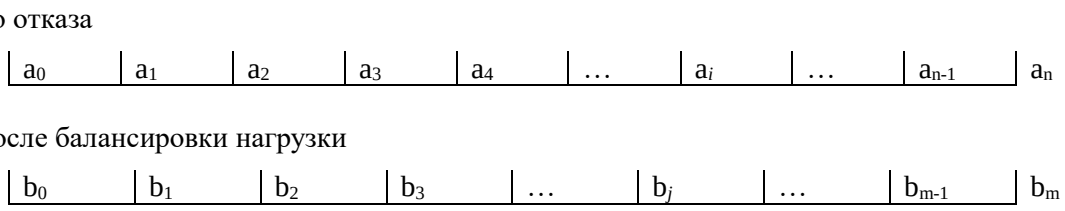

Рис.2. Схема одномерного распределения узлов сетки между процессами до отказа и после балансировки

Сложность восстановления расчетов с балансировкой нагрузки связана с тем, что до отказа распределение данных происходило с одним объемом, а после восстановления MPI объем данных для процессов увеличился. Значит, каждому процессу придется считывать не одну, а несколько контрольных точек. Для заполнения (b_j, b_{j+1}) данными последней контрольной точки необходимо определить, где хранятся необходимые данные, если они в оперативной памяти других процессов, то надо определить у каких.

При сохранении первого уровня каждый процесс i хранит (a_i, a_{i+1}) в оперативной памяти и надо определить, кому какую часть данных нужно посылать. Сохранение второго уровня происходит в распределенную файловую систему и для заполнения данных (b_j, b_{j+1}) не надо локальных пересылок данных, но необходимо чтение нужного файла из распределенной файловой системы.

Определение адресатов передачи от i*	Определение адресатов приема для j*
For j=0:m-1 If $b_j < a_{i*} <= b_{j+1}$ или $b_j <= a_{i*+1} < b_{j+1}$ Включить j в список передачи	For i=0:n-1 If $a_i < b_{j*} <= a_{i+1}$ или $a_i <= b_{j*+1} < a_{i+1}$ Включить i в список приема

Рис. 3 Алгоритм определения адресатов передачи и приема данных

Шаг 1.Восстановление среды MPI.
Шаг 2.Процессы пережившие отказ определяют объем новой нагрузки.
Шаг 3.Определяются адресаты передачи и адресаты приема (см. рис. 3). Дополнительно рассматриваются адресаты для данных соответствующие отказавшим процессам.
Шаг 4. Осуществляется заполнение данными из последней контрольной точки.

Рис. 4. Алгоритм восстановления расчетов, основанный на диффузной балансировке

Например, при отказе первого уровня процесс с номером i=3 хранит (a_3, a_4) и он должен их послать процессам с номерами j=2 и j=3. Например, процесс с номером j=2 должен получить часть данных не только (a_3, a_4), но и (a_2, a_3), то есть при отказе первого уровня j=2 должен получить данные от i=2, i=3 (если они не были в списке отказавших).

Алгоритм определения адресатов передачи и приема данных заключается в рассмотрении областей пересечения сетки и представлен на рис. 3. Аналогичным образом проводится определение адресатов для данных соответствующих отказавшим процессам.

Таким образом, после детекции отказа, общий алгоритм восстановления расчетов, основанный на диффузной балансировке, будет иметь вид см. рис. 4.

Замечание 2. Приведенный алгоритм определения адресатов для рассматриваемой задачи распределения тепла в тонкой прямоугольной пластине имеет такой вид, так как балансировка нагрузки проходила по одной координате. В общем случае определение адресатов осуществляется за счет обхода логически соседних процессов.

3.2 Теоретическая оценка накладных расходов при фиксированном числе отказов

В формуле (1) значение констант C_i, R_i, D постоянно в течение расчета, однако при восстановлении методом балансировки после отказов уменьшается число процессов, и изменяются значения этих констант. Таким образом, формула (1) для двухуровневой стратегии обеспечения отказоустойчивости примет следующий вид

$$T_{\text{итоговое}} = T_{\text{б}} + \sum_k \left(C_1^k\left(x_1^k - 1\right) + C_2^k(x_2^k - 1)\right) + \\ + \frac{T_e}{2x_1^*}n_1 + \sum_k \left(D^k + R_1^k\right) + \frac{T_e}{2x_2^*}n_2 + \sum_k \left(\frac{C_1^k x_1^k}{2x_2^*} + D^k + R_2^k\right) \quad (2)$$

В этой формуле значения C_i^k, R_i^k, D^k, зависят от числа процессов оставшихся после отказа, x_i^k – равно числу периодов между контрольными точками до следующего отказа, а x_i^* – зависит от выбранной частоты отказов и не зависит от числа оставшихся процессов. Вычисление формулы (2) происходит итерационно в зависимости от числа отказов первого и второго уровня. В вычислительном эксперименте расчет проходит при числе процессов от 12 до 36, для каждого этого значения были определены соответствующие коэффициенты C_i^k, R_i^k, D^k формулы (2).

4. Вычислительный эксперимент

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. Реализация отказа заключается в вызове функции raise(SIGKILL).

Были реализованы две параллельные программы «запасные процессы» и «балансировка нагрузки» с двухуровневым обеспечением отказоустойчивости. В первой восстановление осуществляется за счет запасных процессов. Во второй – методом диффузной балансировки нагрузки. Были проведены теоретические оценки накладных расходов согласно параграфам 1.3 и 2.2. Для сравнения работы программ число отказов варьировалось от 1 до 12, причем случаи с разным числом отказов второго уровня рассматривались отдельно. Для выбранного числа отказов p, каждый следующий отказ происходит через одинаковый промежуток времени $1/p$ часов.

Для равенства объема работ выполняемых программами с разными методами восстановления, принято, что они должны выполнить одинаковое число итераций. За эталонное число итераций, берется число итераций работы «чистой» программы, в которой не происходит сохранения контрольных точек, а во время ее работы не происходит отказов. Число процессов, на которых запускается «чистая» программа для первого и второго примера равно 24, для третьего примера - 12, а общее время счета «чистой» программы должно составлять 1 час.

На рис. 5-10 по оси абсцисс отложено общее число отказов в данном расчете, а по оси ординат — доля накладных расходов.

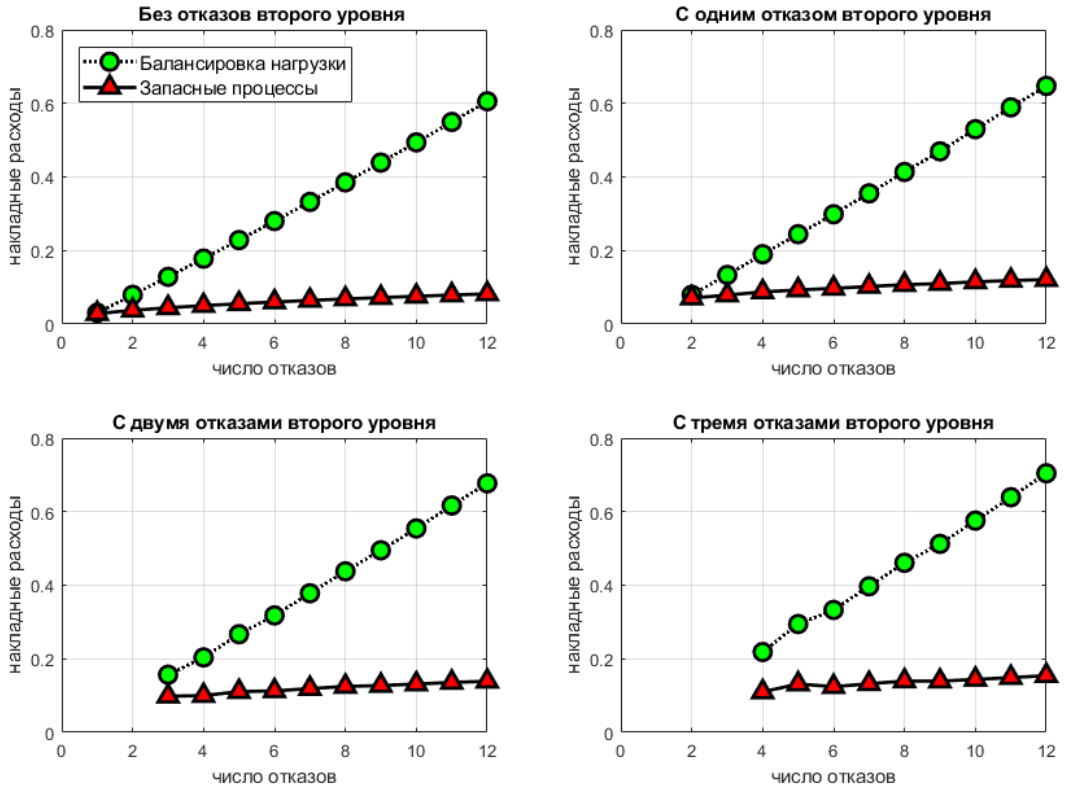

Рис. 5. Теоретическая оценка накладных расходов $N_w = 24$, $N_s = 12$, $N_b = 24$

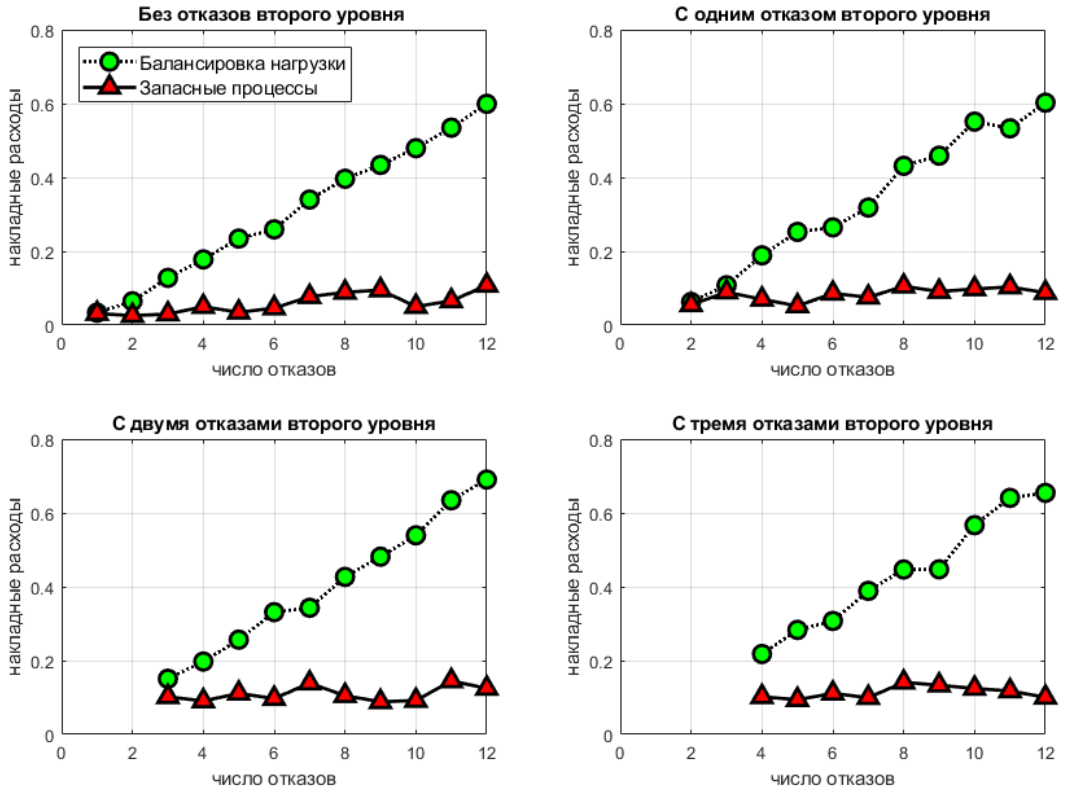

Рис.6. Экспериментальная оценка накладных расходов $N_w = 24$, $N_s = 12$, $N_b = 24$

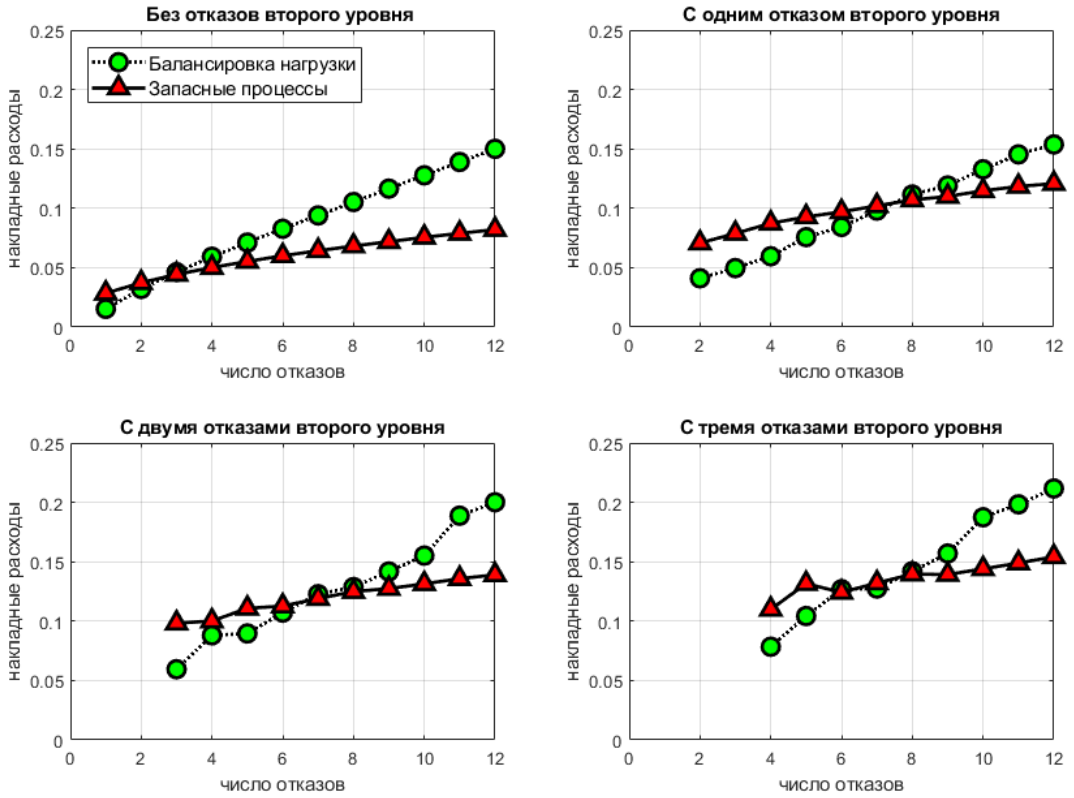

Рис. 7 Теоретическая оценка накладных расходов $N_w = 24$, $N_s = 12$, $N_b = 36$

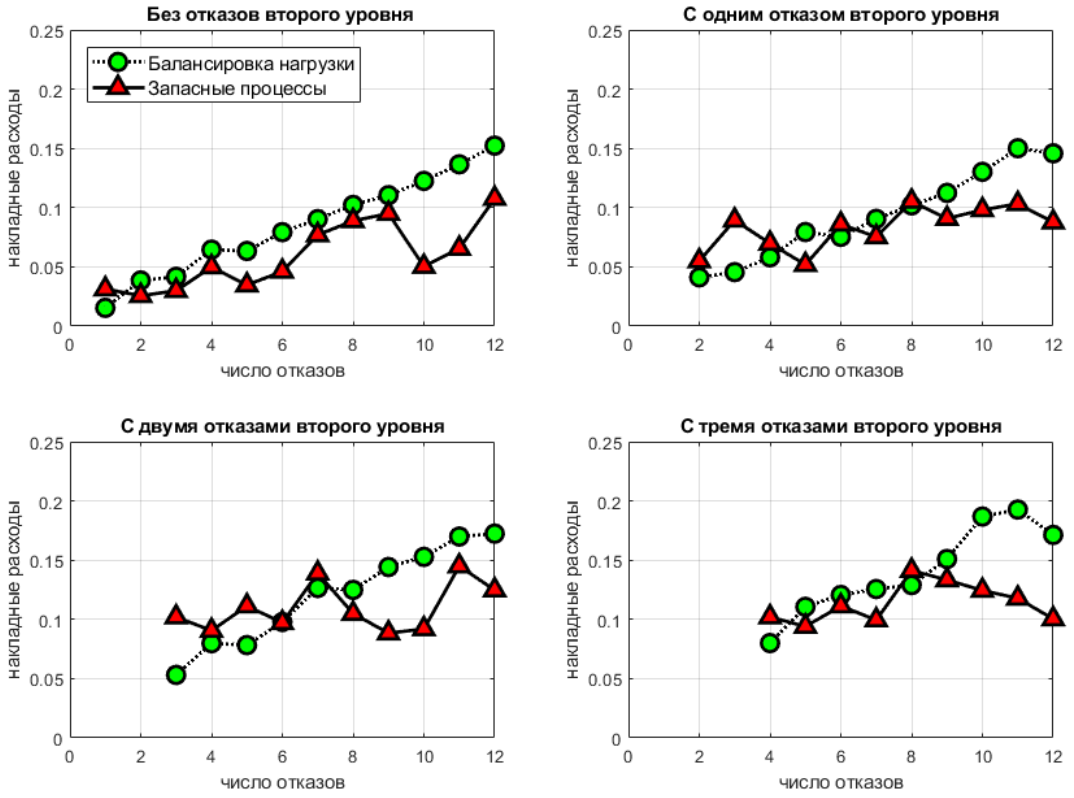

Рис. 8 Экспериментальная оценка накладных расходов $N_w = 24$, $N_s = 12$, $N_b = 36$

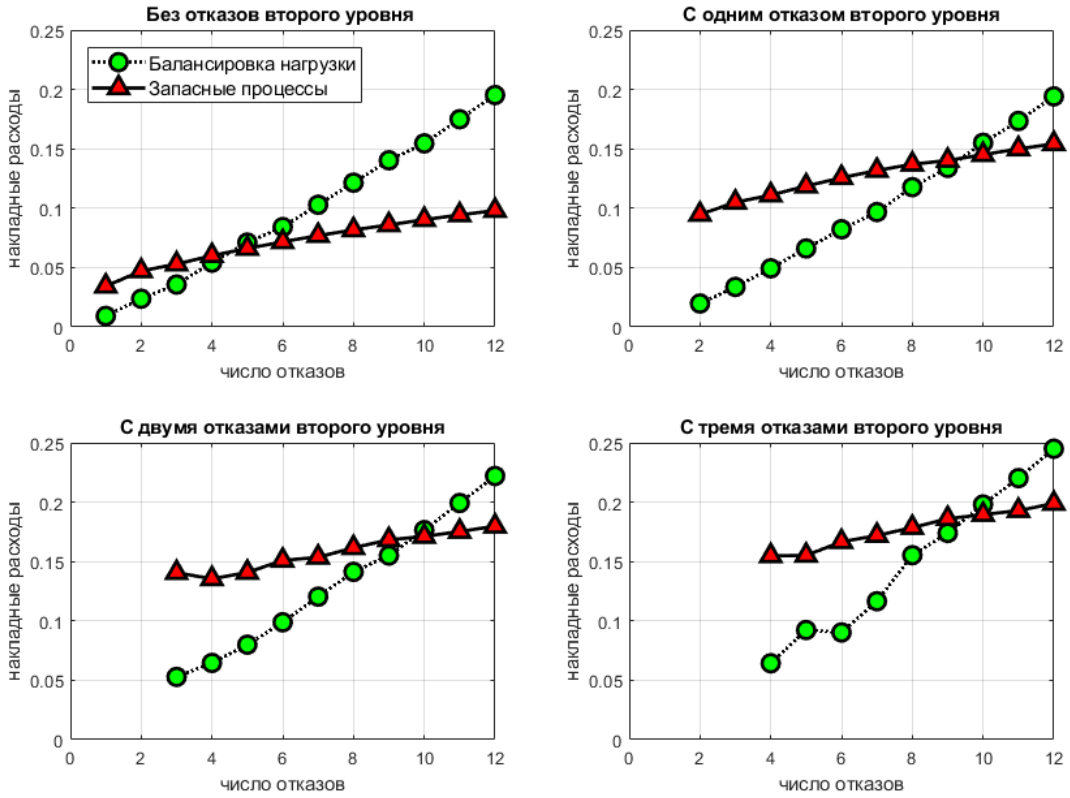

Рис. 9 Теоретическая оценка накладных расходов $N_w = 12$, $N_s = 12$, $N_b = 24$

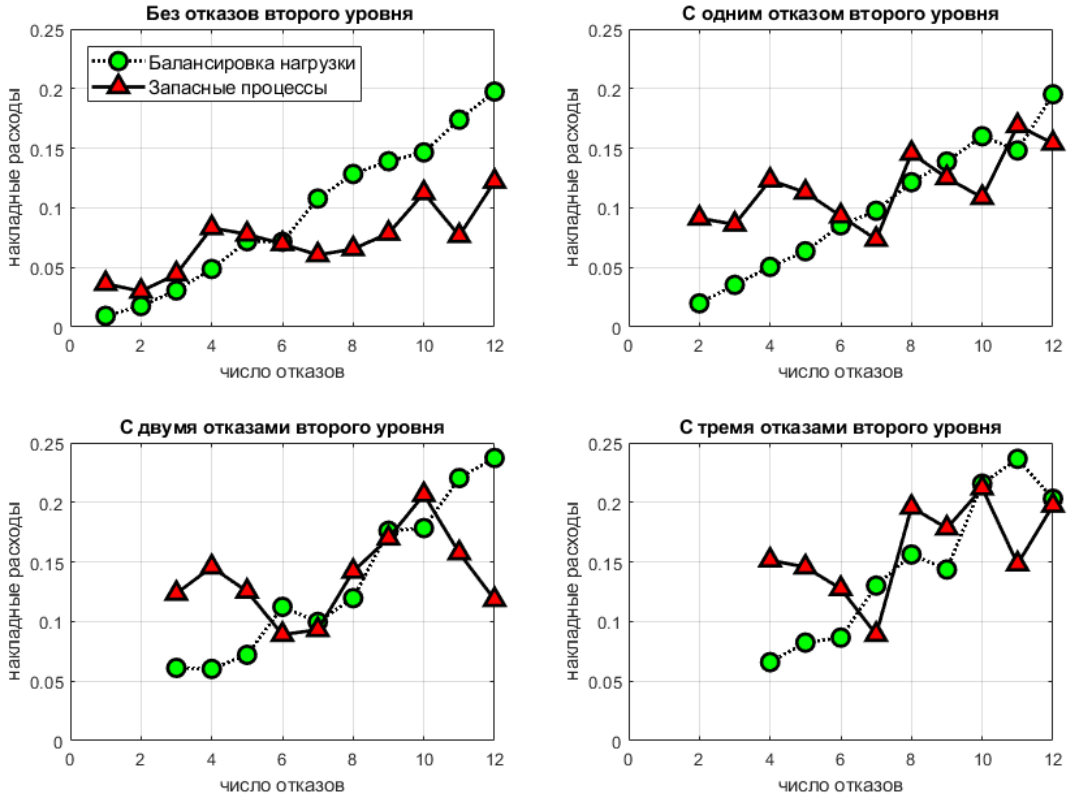

Рис. 10 Экспериментальная оценка накладных расходов $N_w = 12$, $N_s = 12$, $N_b = 24$

В первом примере рис. 5, 6 рассматриваются программа «запасные процессы» с $N_w = 24$ (рабочих процессов) и $N_s = 12$ (запасных процессов) по сравнению с программой «балансировка нагрузки» $N_b = 24$ (процесса). Такие условия взяты с целью начального равенства объема данных для одного процесса каждой программы. Как и ожидалось, накладные расходы второй программы при теоретической и экспериментальной оценке оказались существенно выше.

Во втором примере рис. 7, 8 рассматриваются программа «запасные процессы» с $N_w = 24$ (рабочих процессов) и $N_s = 12$ (запасных процессов) по сравнению с программой «балансировка нагрузки» $N_b = 36$ (процесса). Такие условия взяты с целью равенства начального числа процессов. Полученные теоретические и экспериментальные оценки показывают, что при малом числе отказов накладные расходы близки и иногда метод балансировки приводит к меньшему значению накладных расходов.

В третьем примере рис. 9, 10 рассматриваются программа «запасные процессы» с $N_w = 24$ (рабочих процессов) и $N_s = 12$ (запасных процессов) по сравнению с программой «балансировка нагрузки» $N_b = 36$ (процесса). Такие условия взяты также с целью равенства начального числа процессов. Полученные теоретические и экспериментальные оценки показывают, что при малом числе отказов метод балансировки приводит к меньшему значению накладных расходов.

Теоретические и экспериментальные оценки накладных расходов при отказах с фиксированным моментом наступления демонстрируют, что при небольшом числе отказов для восстановления выгоднее использовать стратегию с балансировкой нагрузки.

5. Заключение

В данной работе рассматриваются две стратегии восстановления расчетов после отказов в системе, а именно с помощью диффузной балансировки нагрузки и стратегии, основанной на выделении группы запасных процессов. Недостатком стратегии балансировки является сложность ее реализации для некоторых алгоритмов, а преимуществом, что все процессы выполняют полезную работу во время счета. Вторая стратегия проста в реализации, так как не требует разработки средств балансировки. Однако запасные процессы фактически простаивают большую часть расчетов, но почти все они выполняют полезную работу к концу расчетов.

Рассматриваемый в работе первый пример показывает, что выделение дополнительных запасных процессов может существенно сократить накладные расходы на отказоустойчивость. Полученные теоретические и экспериментальные оценки накладных расходов в примерах два и три, демонстрируют что, при небольшом числе отказов для восстановления выгоднее использовать метод диффузной балансировки нагрузки.

Эффективность применения метода диффузной балансировки нагрузки при восстановлении расчетов, будет зависеть от особенностей конкретного вычислительного алгоритма и параметров вычислительной системы. Таким образом, перед реализацией балансировки, лучше проводить дополнительное теоретическое исследование эффективности балансировки.

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. 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. 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, 2015. 320 p. DOI: 10.1007/978-3-319-20943-2.

5. 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.

6. Бондаренко А.А., Якобовский М.В. Моделирование отказов в высокопроизводительных вычислительных системах в рамках стандарта Mpi и его расширения ULFM // Вестник Южно-Уральского государственного университета. Серия "Вычислительная математика и информатика". 2015. Т. 4, № 3. С. 5–12.

7. Бондаренко А.А., Ляхов П.А., Якобовский М.В. Координированное сохранение контрольных точек с журналированием передаваемых данных и асинхронное восстановление расчетов после отказов. Суперкомпьютерные дни в России: Труды международной конференции (24-25 сентября 2018 г., г. Москва). – М.: Изд-во МГУ, 2018. С. 694–704.

8. Четверушкин Б.Н., Якобовский М.В. Вычислительные алгоритмы и архитектура систем высокой производительности // Препринты ИПМ им. М.В. Келдыша. 2018. № 052. С. 1–12.

9. Elnozahy E. N., Plank J. S. Checkpointing for peta-scale systems: A look into the future of practical rollback-recovery // IEEE Transactions on Dependable and Secure Computing. 2004. Vol. 1, No. 2. P. 97–108.

10. Oldfield R. A., Arunagiri S., Teller P. J., Seelam S., Varela M. R., Riesen R., Roth P.C. Modeling the impact of checkpoints on next-generation systems. In 24th IEEE Conference on Mass Storage Systems and Technologies. Sept. 2007, P. 30–46.

11. Bland W. et al. Post-failure recovery of MPI communication capability: Design and rationale // The International Journal of High Performance Computing Applications. 2013. Vol. 27, No. 3. P. 244–254. DOI: 10.1177/1094342013488238.

12. Fault Tolerance Research Hub URL: http://fault-tolerance.org/ (дата обращения: 01.12.2018)

13. Sorin D. Fault Tolerant Computer Architecture. Synthesis Lectures on Computer Architecture Morgan&Claypool, 2009. 104 p. DOI: 10.2200/S00192ED1V01Y200904CAC005.

14. 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.

15. 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.

16. 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.

17. Engelmann, C., Vallee, G.R., Naughton, T., Scott, S.L. Proactive fault tolerance using preemptive migration // 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.

18. 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.

19. Di S., Bouguerra M.S.; Bautista-Gomez L., Cappello F. Optimization of multi-level checkpoint model for large scale HPC applications // Parallel and Distributed Processing Symposium, 2014 IEEE 28th International. May 19-23, 2014. Phoenix, USA P. 1181–1190. DOI: 10.1109/IPDPS.2014.122.

20. Якобовский М. В. Введение в параллельные методы решения задач // М.: МГУ. 2013. 328с.