ISSN 2079-3316 ПРОГРАММНЫЕ СИСТЕМЫ: ТЕОРИЯ И ПРИЛОЖЕНИЯ № 3(7), 2011, с. 17–28

УДК 004.052.3

Е. О. Тютляева, А. А. Московский

Методы обеспечения отказоустойчивости в библиотеке шаблонных классов C++ для распараллеливания T-Sim

Аннотация. В работе рассматриваются проблемы отказоустойчивости параллельных приложений при работе на распределенных вычислительных установках. Увеличение масштабов современных вычислительных систем приводит к увеличению вероятности отказа отдельных элементов системы. В ряде случаев вычислительные алгоритмы, такие как генетические алгоритмы, алгоритмы, использующие метод Монте–Карло и т.п., допускают возможность отказа одного или нескольких элементов. В статье предлагаются методы для реализации таких алгоритмов и обеспечения их работоспособности при условии программных и аппаратных сбоев на вычислительных узлах. В рамках библиотеки C++ T-Sim разработан пример редукционного (монотонного) объекта, а также нескольких альтернативных механизмов перезапуска заданий. Проведены пробные отказоустойчивые реализации алгоритма.

Ключевые слова и фразы: Отказоустойчивость, локальная синхронизация, монотонный объект, распределенные системы, шаблонные классы C++.

1. Введение

Интенсивный рост современных высокопроизводительных технологий влечет за собой увеличение масштабов вычислительных систем, что в свою очередь связано с беспрецедентным увеличением количества аппаратных компонентов вычислительной установки. Уже сегодня, по данным рейтинга ТОП–500 [1], лидирующий суперкомпьютер мира Tianhe-1A состоит более чем из семи тысяч узлов, каждый из которых оснащен процессорами общего назначения, графическими ускорителями. Увеличение количества компонентов системы естественным образом ведет к увеличению статистической вероятности отказа отдельных элементов системы и к уменьшению времени

Работа выполнена в рамках программы 13 фундаментальных исследований президиума РАН.

наработки на отказ всей системы в целом. Значительная часть существующего сегодня научного программного обеспечения не поддерживает корректную обработку программных и аппаратных сбоев на вычислительных компонентах. Основная причина заключается в том, что данное программное обеспечение изначально проектировалось для работы на вычислительных установках, состоящих из десятков узлов, в связи с чем вероятность аппаратного сбоя во время работы задачи не была критично высокой. Следовательно, проектирование отказоустойчивых алгоритмов с неизбежными накладными расходами не представлялось целесообразным. В современных реалиях, при увеличении масштабов как современных вычислительных систем, так и объемов вычислений для задач, стоящих у переднего края науки, обеспечение отказоустойчивости вычислительных приложений является одним из основных приоритетов. Это связано с тем, что вероятность отказа одного из множества используемых компонент системы во время работы вычислительной задачи очень высока. Под отказом может подразумеваться сбой коммуникационной подсети, выход из строя аппаратного или даже программного компонента (к примеру, аварийное завершение сервиса ssh на узле может привести к тому, что полностью исправный узел будет абсолютно не доступен приложениям, использующим данный протокол). В настоящее время существует несколько подходов к организации отказоустойчивой работы на вычислительных установках больших масштабов. Прежде всего, это повышение надежности установки на аппаратном уровне, дублирование основных управляющих компонентов, поддержка «горячей замены» отказавшего оборудования и т.п. Отказоустойчивость на уровне программного обеспечения представляется более экономически выгодным решением, в связи с тем, что корректно реализованный отказоустойчивый алгоритм в режиме нормальной работы оборудования может использовать все доступные ресурсы для получения максимальной производительности, а при выходе из строя одного из компонентов (узлов) продолжить вычисления на оставшихся узлах. Реализация подобных отказоустойчивых алгоритмов требует разработки нового подхода к организации распределенных вычислений. В статье «Развитие отказоустойчивого программного обеспечения» [2] данная идея сформулирована следующим образом: «нам, сообществу разработчиков программного обеспечения, необходимо перейти от парадигмы стабильных и неизменных аппаратных ресурсов

с глобальной синхронизацией процессов к локально синхронизируемым, динамически планируемым и иерархически управляемым приложениям, которые смогут завершить вычисления несмотря на ожидаемое количество отказов аппаратных компонент». В своей работе мы предлагаем пользователю инструментарий, позволяющий разрабатывать алгоритмы для ряда задач в рамках предложенной парадигмы. В частности, мы попытались предложить и реализовать в виде элементарных конструкций языка параллельного программирования несколько методов для обеспечения отказоустойчивой работы ряда алгоритмов при условии программных и аппаратных сбоев.

2. Отказоустойчивые алгоритмы

На уровне программного обеспечения также можно выделить ряд подходов к организации отказоустойчивых вычислений. Прежде всего, это получивший широкое распространение подход с сохранением контрольных точек состояния вычисления [3, 4] и восстановлению до ближайшей доступной контрольной точки в случае сбоя. Можно выделить несколько стратегий, в том числе с сохранением контрольных точек в центральном хранилище, на локальных дисках вычислительных узлов, и, в конце концов, бездисковое сохранение контрольных точек. Существенным недостатком такого подхода являются очень высокие накладные расходы. По некоторым подсчетам [5], время, затрачиваемое на сохранение контрольных точек и восстановление вычисления, может превысить время наработки на отказ для суперкомпьютеров, которые будут занимать лидирующие позиции к 2015 году. С этой точки зрения более перспективными представляются адаптивные подходы, которые при обнаружении сбоя в процессе вычисления выполняют необходимые действия, направленные на продолжение вычисления на оставшихся в установке работоспособных узлах без аварийного завершения работы всей программы.

К сожалению, такой подход применим не ко всем задачам, так как сбои могут привести к полной потере подзадач, которые вычислялись на аварийном узле. Тем не менее, существует ряд отказоустойчивых алгоритмов, для которых можно естественным образом реализовать стабильную работу в вычислительном окружении с высокой вероятностью отказа компоненты. К таким алгоритмам относятся генетические алгоритмы, алгоритмы, реализующие методы Монте–Карло, и

другие аналогичные алгоритмы. В статье «Разработка отказоустойчивых по природе алгоритмов для вычисления на 100,000 процессорах» [6] такие алгоритмы называются «отказоустойчивыми по природе» и определяются следующим образом: «мы будем называть отказоустойчивыми по природе такие алгоритмы, которые могут завершить вычисления с получением корректного ответа несмотря на сбой в работе одной или нескольких вычислительных подзадач по время вычисления». Авторы данной работы предложили метод для отказоустойчивого вычисления глобального максимума и конечной разницы и испытали его на установке, симулирующей работу установки с 100,000 узлами и значительным количеством общих отказов. Идея предложенного алгоритма заключалась в том, что как только один из вычислительных узлов находит локальное значение (к примеру, локальный максимум), он тут же транслирует его соседям, что позволяет сохранить найденное решение даже в случае отказа.

Другой подход, который применим для отказоустойчивых по природе задач, это повторная посылка подзадачи в случае сбоя. Такая техника достаточно широко используется в распределенных вычислениях, особенно в вычислительных сетях, к примеру SETI@HOME [7] и FOLDING@HOME [8], где единственной точкой отказа является центральный сервер, рассылающий задания. Для центрального сервера можно обеспечить отказоустойчивость на аппаратном уровне. Отказ любого другого компьютера, участвующего в расчетах, компенсируется при помощи выполнения центральным сервером переназначения задачи другому компьютеру.

В данной работе мы предлагаем подход, который представляет собой модификации двух предыдущих методов, а также реализацию этого подхода в виде конструкций языка параллельного программирования высокого уровня.

3. Обработка сбоев на уровне языка T-Sim

Как уже было подчеркнуто во введении, обеспечение отказоустойчивости требует специальной парадигмы распределенных вычислений. Как неоднократно отмечалось на мировых конференциях и семинарах по распределенным вычислениям, наиболее перспективными в этом свете являются языки, которые поддерживают локально контролируемые объекты для легковесной синхронизации и контроля состояния распределенного вычисления, включая параллелизм по

данным и неготовые переменные[1] [9, 10]. В связи с этим более перспективным представляется язык параллельного программирования, который будет поддерживать объекты с локальной синхронизацией.

На основании данных теоретических рассуждений для реализации моделей отказоустойчивости был выбран язык параллельного программирования T-Sim [11]. Это язык параллельного программирования, поддерживающий распределенные вычисления и реализующий параллелизм по задачам. Основными идеями, заложенными в этот проект, являются бесконфликтные вычисления на основе чистых (не имеющих побочных эффектов) функций и неготовых переменных для синхронизации вычислений. Данная парадигма соответствует требованиям к языку программирования, на котором можно создавать отказоустойчивые приложения. Дополнительными особенностями библиотеки является использование возможности конструирования дополнительных стратегий планирования вычислений, создание специализированных типов данных в виде классов–наследников от базовых примитивов, т.к. библиотека реализована в виде шаблонных классов C++.

Добавление поддержки отказоустойчивости в библиотеку T-Sim потребовало изменений на нескольких уровнях. Прежде всего, нужно было добавить корректную обработку сбоев на вычислительных узлах в базовую систему назначения и отправления заданий. Для этого изучен и модифицирован стандартный процесс обработки и посылки сообщения.

Под сообщением в T-Sim понимается любой примитив, который может быть отправлен на другой вычислительный узел. Это может быть и вычислительная подзадача, и надлежащим образом конвертированные данные, и короткое коммуникационное сообщение. Схематически процесс обработки сообщений изображен на рис. 1. Обычными линиями изображена стандартная схема обработки, пунктирные линии схематически отражают модификации в системе обработки сообщений. Те сообщения, для которых по логике программы строго определен узел назначения, такие как, к примеру, коммуникационные сообщения, имеют вид «сообщение->узел_назначения» и ставятся непосредственно в глобальную очередь сообщений для отправки.

[1] В англоязычной литературе часто используется термин futures

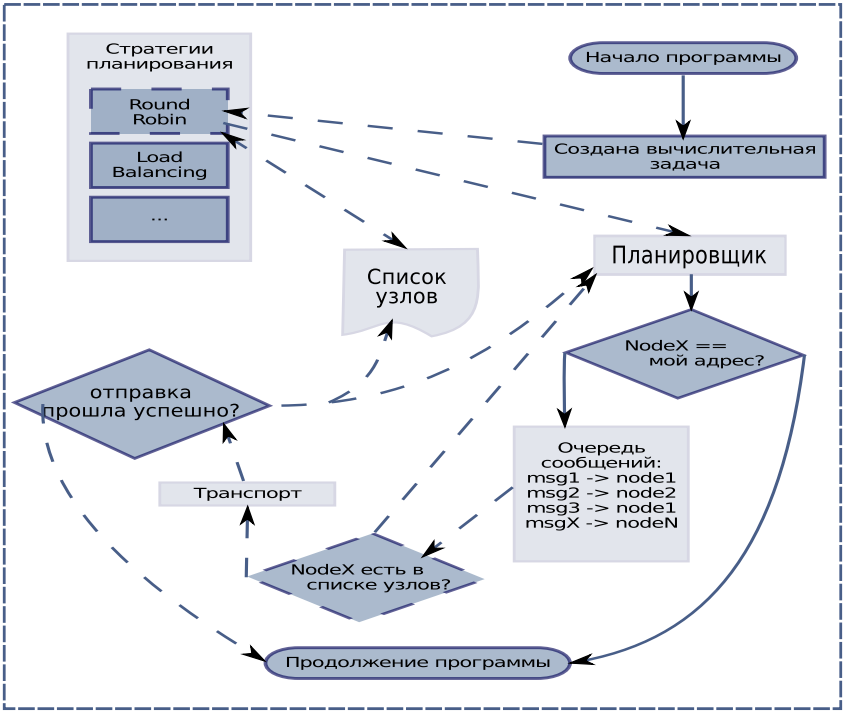

Рис. 1. T-Sim

Вычислительные подзадачи сперва обрабатываются при помощи планировщика, который для каждого элемента запрашивает специализированный метод стандартной (либо определенной пользователем) стратегии планирования.

Запрошенный метод определяет, будет ли задача вычисляться на данном узле, или будет перенаправлена на другой вычислительный узел из списка участвующих в вычислении узлов, который формируется при запуске программы при помощи специального конфигурационного файла, где описаны все участвующие в вычислении узлы. Если задача назначена на выполнение на данном узле, она выполняется, в противном случае формируется объект — сообщение общего вида, которое содержит вычислительную подзадачу и адрес узла назначения, и данное сообщение также добавляется в глобальную очередь сообщений для отправки.

В процессе выполнения задачи извлекаются из глобальной очереди планирования для отправки и отправляются. Если отправление сообщения не проходит успешно, выполнение программы аварийно завершается.

Такая система функционирует на каждом узле, что обеспечивает необходимую локальную динамическую синхронизацию работы всей системы.

В рамках данной работы описанная выше система была модифицирована для обеспечения корректной обработки аварийного сбоя в вычислительном окружении.

В модифицированной версии целевой узел считается аварийным если отправление сообщения не проходит успешно. Признанный аварийным узел удаляется из списка участвующих в вычислении узлов, а сообщение, во время отправления которого произошел сбой, отправляется обратно в очередь планирования для переназначения целевого узла. Модифицированная версия планировщика работает с актуальной версией списка доступных узлов, что позволяет избежать назначения новых задач на отказавший узел после обнаружения сбоя. В группе риска оказываются те задачи из глобальной очереди планирования на отправку, для которых узел назначения был определен до обнаружения сбоя. Данные задачи могли быть запланированы на выполнение на аварийном узле. В связи с этим проводится дополнительная проверка (см. пунктирные линии на рис. 1). После проверки сразу выполняется переназначение целевого узла без попытки повторной отправки, чтобы избежать возможного ожидания таймаута от аварийного узла.

Такой подход позволяет оперативно единообразно обрабатывать различные отказы в вычислительном окружении, включая отказы сети, программного и аппаратного обеспечения вычислительных узлов, с минимальными накладными расходами.

4. Повышение отказоустойчивости

Гораздо большую сложность представляет обработка задач, которые были направлены на отказавший узел, а во время вычисления на узле случился сбой. Система T-Sim продолжает ожидать результат вычисления от данного узла, подобно большинству распространенных систем программирования, и технически программа не может корректно завершиться даже для тех задач, для которых получение ответа для всех подзадач не критично.

Мы предлагаем два варианта, оба из которых применимы только для задач, которые отказоусточивы по природе.

В рамках данной работы создан отдельный тип для таких задач — неготовая переменная с заданным временем жизни. Этот тип обладает всеми свойствами неготовой переменной, для него определена сериализация, что позволяет корректно оперировать переменными данного типа в распределенной памяти. Но, в отличие от обычной неготовой переменной, в шаблонных параметрах данного типа можно указать еще и время жизни такой переменной. Создание переменной такого типа проводится аналогично созданию обычной неготовой переменной в T-Sim (аналогично созданию экземпляра шаблонного класса в терминах C++), к примеру: TValT<int, 10> var. Созданная таким образом неготовая переменная var будет иметь значение типа int и время жизни, равное 10 секундам. Добавление концепции времени жизни означает, что после истечения времени жизни результат такой переменной ожидаться не будет и программа будет корректно завершена, даже если результат вычисления каких-то функций, ожидаемых в данной переменной, не получен. В отличие от обычной неготовой переменной, которая имеет 2 состояния: готова (и тогда значением данной переменной является вычисленное значение) и не готова, неготовая переменная со временем жизни имеет третье, дополнительное состояние — результат по прежнему не вычислен, а время жизни истекло. Программист имеет возможность обрабатывать полученный результат согласно логике задачи, т.е. он может оставить данный результат невостребованным, запустить аналогичную или перезапустить эту же задачу с ожиданием результата в другой неготовой переменной со временем жизни, при этом увеличив время жизни и т.п. Определять время жизни программист также может как эвристически, так и программно оценив среднее время работы аналогичных подзадач. Такой подход позволяет получить результат за детерминированное время для тех задач, которые позволяют получить корректный результат (возможно, с меньшей точностью), даже при условии, если одна или несколько вычислительных подзадач не завершили свою работу. Достоинством такого подхода является гарантированная завершаемость при любом количестве и типе отказов на вычислительных узлах, простота применения и минимальные накладные расходы.

Недостатком такого подхода является сложность применения в случае неравновесных гранул параллелизма, которые могут, к примеру, быть порождены при рекурсивном вычислении. Поэтому был разработан метод сохранения задач.

Технически, для реализации сохранения задач был разработан тип–наследник от базового типа, описывающего вычислительные задачи, и для этого типа определено запоминание задач. На каждом узле, в локальном экземпляре планировщика создается приватный объект, позволяющий оперировать списком сохраненных задач вида std::map <узел, список назначенных задач>. Данный объект тоже локален, т.е. на каждом узле определен локальный экземпляр списка сохраненных задач, в котором сохранены все подзадачи, созданные на данном узле и направленные для обработки на другие узлы из вычислительного окружения. Вычислительная задача, для которой определено сохранение, добавляется в этот список после того, как планировщиком для нее определен узел назначения. После завершения вычисления задача такого типа отсылает отчет о завершении на родительский узел. На родительском узле, после получения такого отчета, завершенная задача удаляется из списка сохраненных.

В случае перехвата отказа одного из вычислительных узлов, все оставшиеся активными узлы удаляют адрес данного узла из списка доступных для проведения вычисления и выполняют повторную посылку тех задач, которые были на него отправлены. Такой подход позволяет обеспечить отказоустойчивость даже для задач с неравновесными гранулами параллелизма, к тому же использование данного метода позволяет довести до конца обработку задач, которые были направлены на отказавший узел, а во время вычисления на узле случился сбой, т.к. после того, как сбой будет обнаружен, задача будет заново перенаправлена на другой узел. Недостатком такого подхода является наличие высоких накладных расходов на сохранение задач и требуемой оперативной памяти на поддержание списка задач. Достоинством является движение в сторону локальной синхронизации, т.к. каждый узел помнит те задачи, для которых было проведено локальное назначение.

Для того, чтобы гарантированно выявлять сбои, а не ждать завершения работы задачи, направленной на аварийный узел, в тех случаях, когда больше никакие задачи на этот узел не направляются и таким образом невозможно выявить сбой в работе узла во время естественного хода вычисления, также реализована дополнительная

стратегия планирования, которая через заданные промежутки времени опрашивает состояние всех участвующих в вычислении узлов. Такой подход позволяет гарантированно определять все аппаратные сбои, произошедшие во время вычисления, и вычислять необходимые подзадачи как минимум для задач, отказоустойчивых по природе.

5. Отказоустойчивый поиск наилучшего результата

В рамках испытания разработанных примитивов на языке T-Sim был реализован монотонный редукционный объект.

Каждый узел, участвующий в вычислениях, поддерживает локальную копию наилучшего найденного результата (к примеру, глобального минимума). Все узлы участвуют в вычислении минимума, при этом каждый узел обрабатывает независимые подзадачи. Если на одном из локальных узлов удается получить более минимальное значение, чем известный данному узлу глобальный минимум, то на данном узле выполняется обновление глобального минимума и формируются сообщения всем участвующим в вычислении узлам, содержащие оповещение о новом рекорде. Данные действия — обновление рекорда и формирование и постановка в очередь рассылки сообщений с обновлением — выполняются в единой транзакции во избежание возможных нежелательных эффектов.

При получении сообщения с обновленным рекордом на вычислительном узле проводится сравнение полученного рекорда с известным данному узлу глобальным минимумом, и если полученное значение меньше, то происходит транзакционное обновление рекорда и оповещение всех соседей, в противном случае сообщение просто игнорируется.

Такая схема позволяет гарантировать распространение полученного рекорда по сети даже в том случае, если узел, обнаруживший рекордное значение, успел уведомить о нем только одного из своих соседей. Также такая схема позволяет избежать конфликтов, если несколько узлов одновременно нашли и начали распространение разных рекордных значений — вне зависимости от порядка распространения рекорда и отказов в системе через определенное количество итераций глобальный минимум, известный каждому узлу, будет обновлен новым рекордным значением.

Тестирование работы данного объекта проводилось на двух вычислительных кластерах, с двумя и четырьмя узлами. Сбои в вычислительном окружении имитировались несколькими способами:

- Принудительное завершение клиентской части программы на одном из вычислительных узлов.
- Принудительная остановка (не завершение) при помощи отладчика gdb клиентской части исполняемой программы на одном из узлов (данный тест имел особенное значения для тестирования работы неготовых переменных с фиксированным временем жизни).
- Отключение сервиса ssh на одном из вычислительных узлов.
- Имитация аппаратного сбоя — отключение сетевого кабеля от одного из вычислительных узлов во время исполнения отказоустойчивого алгоритма.

Данные методы имитации сбоев, в различных комбинациях, проводились на одном или нескольких вычислительных узлах, на которых функционировал монотонный редукционный объект и проводился поиск рекордных значений. Запущенная программа определяла сбой, удаляла проблемный узел из списка узлов и продолжала корректно функционировать на оставшихся узлах. Рекорды, полученные на аварийном узле и переданные хотя бы одному из участвующих в вычислении узлов, были успешно распространены по всему окружению.

Подобная схема обладает очень высокой природной отказоустойчивостью и способна получить и сохранить корректный результат на установке сколь угодно большого масштаба.

6. Заключение

В данной работе были рассмотрены методы организации отказоустойчивых вычислений для ряда алгоритмов, отказоустойчивых по природе. Был проведен детальный обзор современных тенденций в области разработки отказоустойчивого программного обеспечения в свете увеличения масштабов ведущих вычислительных систем. Были сформулированы и реализованы для языка параллельного программирования T-Sim основные примитивы отказоустойчивости. Комбинация языка параллельного программирования T-Sim и реализованных методов отказоустойчивости способствует переходу к концепциям локальной синхронизации, динамического планирования, и следовательно, корректной завершаемости вычислительных приложений в нестабильном вычислительном окружении. Приведен пример отказоустойчивого алгоритма, рассчитанного на корректную работу на многокомпонентных нестабильных вычислительных системах.

Список литературы

[1] Рейтинг ТОП–500, Ноябрь, 2010, http://www.top500.org/lists/2010/11. ↑1

[2] Baxter D. *Developing Fault-tolerant Software. A shift in design paradigms is needed to accommodate hardware failure* // Scientific Computing, 2010. ↑1

[3] Plank J. S., Li K. *Faster checkpointing with $n+1$ parity* // FTCS, 1994, p. 288–297. ↑2

[4] Plank J. S., Li K., Puening M. A. *Diskless checkpointing* // IEEE Trans. Parallel Distrib. Syst., 1998. **9**, no. 10, p. 972–986. ↑2

[5] Cappello F., Geist A., Gropp B., Laxmikant V. K., Kramer B., Snir M. *Toward Exascale Resilience* // International Journal of High Performance Computing Applications, November, 2009. **23**, p. 374–388. ↑2

[6] Geist A., Engelmann C. *Development of Naturally Fault Tolerant Algorithms for Computing on 100,000 Processors* // Parallel Computing, 2002, http://www.csm.ornl.gov/~geist/Lyon2002-geist.pdf. ↑2

[7] *Search for ExtraTerrestrial Intelligence at home*. University of California, 2011, http://setiathome.berkeley.edu/. ↑2

[8] *Folding@home*. Vijay Pande and Stanford University, 2010, http://folding.stanford.edu/. ↑2

[9] Sterling T. *Enabling Exascale Computing through the ParalleX execution model* // ECMWF Workshop, 2010. ↑3

[10] Sterling T. *An Overview of Exascale Architecture Challenges* // SC08 Workshop, 2008. ↑3

[11] Московский А. А. *T-Sim — библиотека для параллельных вычислений на основе подхода T-системы* // Международная конференция «Программные системы: теория и приложения» : Наука-Физматлит, октябрь 2006, с. 183-193. ↑3

E. O. Tyutlyaeva, A. A. Moskovskii. *T-Sim fault tolerance.*

Abstract. This paper addresses fault-tolerance challenges in distributed computing environment. Increasing scalability of modern computational clusters leads to an increasing probability of an interrupt occuring. In a number of cases computational algorithms, such as genetic algorithms, Monte Carlo based algorithms, have the mathematical properties that they get the correct answer despite the occurrence of faults in the system. This paper proposes methods for implementation such class of algorithms despite software and hardware faults. Some example of monotonous reducing object is implemented using C++ template class library T-Sim. Moreover, some test realizations are implemented.

Key Words and Phrases: Fault-tolerance, T-Sim C++ template library, monotonous object, local synchronization.

Образец ссылки на статью:

Е. О. Тютляева, А. А. Московский. *Методы обеспечения отказоустойчивости в библиотеке шаблонных классов C++ для распараллеливания T-Sim* // Программные системы: теория и приложения : электрон. научн. журн. 2011. № 3(7), с. 17–28. URL: http://psta.psiras.ru/read/psta2011_3_17-28.pdf