Открытая Т–система: распределённые вычисления в Internet[1]

А.В. Инюхин

Аннотация. В настоящей публикации рассмотрены возможности технологии автоматического динамического распараллеливания, реализованные в новой версии Т–системы [13, 3], в открытой Т–системе (или OpenTS), для выполнения распределённых вычислений в среде Internet, а также представлены результаты экспериментов, иллюстрирующие перспективы подобных вычислений.

1. Введение

Доминирующим в создании вычислительных систем высокой и сверхвысокой производительности в последние годы стал подход, объединяющий в рамках одной системы с помощью высокоскоростной внутренней сети большое количество многопроцессорных узлов. Сетевая среда на базе различных технологий и протоколов позволяет объединять такие, как их называют, сильносвязанные кластеры в слабосвязанные метакластерные структуры. Среда, обеспечивающая связность отдельных вычислительных кластеров в метакластер, может быть самой различной — от традиционной сети Internet и виртуально выделенных каналов ограниченной пропускной способности до высокоскоростных сетей, построенных на технологиях следующего поколения. Подобные структуры эффективны по соотношению цены и производительности, а также по целому ряду других показателей, причин и перспективных сторон их использования.

С учётом вышеизложенного интерес к исследованиям, связанным с подходом к созданию и с оценками эффективности эксплуатации распределённых информационно–вычислительных комплексов в последние годы в мире постоянно растёт.

К высокопроизводительным инструментальным средствам, обеспечивающим построение и сопровождение метакластерных структур на сегодня, например, можно отнести Condor [6], Globus Toolkit [7], к средствам более низкого уровня IMPI [8], MPICH [10], а также ряд других. Уровень апробации каждого из названных средств различен и сегодня на повестке дня остро стоит вопрос исследования и анализа их эффективности, создания новых средств подобного назначения.

[1] Механико–математический факультет МГУ им. М. В. Ломоносова.

В работе, результаты которой представлены в настоящей публикации, использованы возможности PACX [12] и открытой Т–системы автоматического динамического распараллеливания программ (OpenTS)для построения на их основе и сопровождения метакластерных структур.

В качестве среды связности отдельных вычислительных кластеров в метакластер рассматривается Internet без каких-либо дополнительных требований к сети.

2. Постановка задачи

Конечной целью исследований, результаты которых представлены в настоящей публикации, было проведение вычислительного эксперимента с использованием традиционного тестового приложения на однородном (с точки зрения аппаратно–программной платформы) метакластере.

Для поддержки распределённых вычислений использовалась технология автоматического динамического распараллеливания программ, реализованная в OpenTS. Технологические решения, используемые в OpenTS, являются перспективными и представляют следующий этап развития идей, заложенных в оригинальную разработку ИПС РАН «Т–система» [13] и затем в GRACE, разрабатываемую ИПС РАН совместно с МГУ [1].

Как и GRACE [3, 4], OpenTS обладает возможностью адаптации к неоднородностям вычислительной среды и ставит одной из своих основных целей возможность эффективной работы в подобных условиях.

В качестве приложения был выбран стандартный вычислительный тест EP (Embarrassingly Parallel) из пакета NPB 2.3 [11]. Выбор данной задачи для проведения экспериментов вызван её высоким параллелизмом (производится генерация и вычисление суммы большого количества независимых векторных величин). Логика решения такой задачи позволяет обеспечить удовлетворительное функционирование системы динамического распараллеливания, использующей парадигму функционального программирования, а также соответствующую потребность в планировании коммуникационных и вычислительных ресурсов.

Для реализации конечной цели автору потребовалось провести детальный анализ особенностей и функциональных возможностей PACX MPI на коммуникационном уровне и при запуске приложений выявить и устранить «узкие места» и преодолеть возникающие при этом трудности. Потребовалась модернизация кода OpenTS и определённая настройка её планировщика.

Данная публикация представляет результаты, полученные на реальной метакластерной установке, вычислительные ресурсы которой размещены не только в разных регионах России, но и за её пределами. Испытания демонстрирует высокие показатели утилизации вычислительных ресурсов для задач, подобных рассматриваемой.

3. Создание метакластера

3.1. Коммуникационный уровень

OpenTS использует коммуникационный уровень на основе интерфейса MPI. Кроме поддержки различных реализаций MPI осуществлена поддержка трёх метакластерных вариантов MPI: LAM/IMPI [5, 9], MPICH–G2 и PACX.

Так как взаимодействие OpenTS с MPI осуществляется через интерфейс DMPI, то описываемые в данной публикации изменения OpenTS независимы от используемой среды выполнения. Тем не менее, существуют некоторые особенности реализации DMPI–драйверов, которые будут рассмотрены ниже, в подразделе 4.1.

В подразделе 3.3 будет подробно рассмотрен вариант запуска с использованием PACX, как представляющий наибольший интерес из-за простоты использования и возможности эффективной утилизации аппаратных ресурсов.

Работа с метакластером на основе LAM/IMPI подробно описана в [4]. Заметим, однако, что LAM 7.0 и выше для работы с OpenTS непригоден из-за отсутствия функции `MPI_Iprobe()` при использовании протокола IMPI. Как следствие, в подобной конфигурации необходимо использовать LAM версий 6.5.7–6.5.9 с поддержкой расширений IMPI.

MPICH–G2, в свою очередь, имеет ряд существенных недостатков, которые также могут создать проблемы при реализации метакластерных вычислений. Наиболее серьёзный из них — необходимость установления прямых TCP–соединений между взаимодействующими вычислительными узлами задачи, что часто оказывается невозможным. Отмеченное обстоятельство делает MPICH–G2 немасштабируемым и значительно снижает область его использования, особенно при независимом администрировании используемых вычислительных ресурсов.

Тем не менее, OpenTS обладает возможностью использовать любой из перечисленных вариантов, а также не исключено, что и другие. В связи с тем, что представленные в настоящей публикации результаты получены без использования дополнительной информации о транспортном уровне, можно ожидать положительных результатов на аналогичных средах исполнения, не производя дополнительных изменений кода приложения и OpenTS.

3.2. Среда PACX MPI

PACX (PArallel Computer eXtension) MPI, как и MPICH–G2, представляет собой надстройку над интерфейсом MPI, обеспечивающую возможность объединения нескольких вычислительных систем в одну, используя эффективную среду выполнения в каждой из них.

Реализация PACX основана на технологиях IMPI, однако традиционно широко используемая версия 4.1.4 не предоставляет данный протокол на внешнем уровне. С учётом этого обстоятельства данная версия не имеет возможности взаимодействия с реализациями, поддерживающими IMPI, такими, как LAM.

Последние версии PACX (5.0–beta от 06.02.2004) уже включают тестовую поддержку IMPI, но в связи с тем, что пока реализованы далеко не все необходимые для работы OpenTS функции, данный вариант не представляет практического интереса. Также следует отметить, что использование протокола IMPI в PACX не избавляет от необходимости явно описывать конфигурацию метакластера, что с одной стороны, позволяет избежать проблем, связанных с неправильным определением адреса для внешнего взаимодействия (для LAM/IMPI это решалось с помощью введения переменной окружения `IMPI_HOST_NAME`, [4]; ожидается, что данная модификация будет включена в LAM версии 7.1), а с другой — создаёт дополнительные трудности при попытке автоматизировать запуск распределённого приложения.

Работа с каждой конкретной реализацией MPI нижнего по отношению к PACX уровня может иметь ряд особенностей. Например, при использовании Myrinet были обнаружены конфликты внутренних имён в библиотеках PACX и MPICH–GM. Для решения проблемы пришлось внести небольшие изменения в исходные тексты PACX, переопределив имена конфликтующих функций.

3.3. Запуск приложения

Каждая часть задачи PACX требует два дополнительных MPI–процесса для обеспечения коммуникаций между вычислительными кластерами (in–server и out–server). Эти коммуникации происходят по протоколу TCP/IP, и, как следствие, требуется возможность установления такого соединения.

В связи с тем, что управляющие машины используемых кластеров не соединены сетью SCI с вычислительными узлами, запуск MPI–процесса на них оказывается невозможен, и необходимо установление соединения между вычислительными узлами кластеров. Однако, в отличие от MPICH–G2, не требуется установление прямых TCP–соединений между всеми взаимодействующими вычислительными узлами, а достаточно соединить только коммуникационные процессы. Например, можно воспользоваться `ssh` с ключом `-L`, что и было проделано в экспериментальных запусках.

Рассмотрим подробнее процесс запуска приложения в PACX–метакластере, состоящем из трёх вычислительных установок с именами внешних сетевых интерфейсов управляющих машин `cluster0`, `cluster1`, `cluster2` и числом узлов 32, 24 и 16 соответственно. Пусть также внутренний интерфейс называется `server-N`, а вычислительные узлы данных установок имеют имена вида `node-N-X`, где `N` — номер кластера, 0–2, а `X` — MPI–ранг узла при выполнении `mpirun`.

В первую очередь, для запуска приложения необходимо создать на каждом из вычислительных кластеров файл `.hostfile` с описанием используемых ресурсов. Каждая строка данного файла состоит из имени узла, обладающего внутренним MPI–рангом 0 и осуществляющего внешние коммуникации, а также количества вычислительных узлов в соответствующем кластере. Порядок перечисления кластеров должен совпадать. В случае соединения через

туннель, вместо имени узла указывается управляющая машина, то есть, `server-N` в рассматриваемом случае. Например, для кластера `cluster2` файл `.hostfile` может выглядеть, как изображено на рис. 1.

```
server-2 30
server-2 22
node-2-0 14
```

Рис. 1. Пример конфигурационного файла `.hostfile` для PACX MPI.

Следующим этапом является установление ssh–туннелей для возможности соединения коммуникационных процессов. В каждом направлении требуется обеспечить два соединения: с процессами, имеющими внутренние MPI–ранги 0 и 1. В рассматриваемом случае туннели устанавливаются на `cluster1` командой, приведённой на рис. 2 и на `cluster2` командами, приведёнными на рис. 3.

```
[cluster1]$ ssh cluster0 -L31000:node-0-0:31000
-L31001:node-0-1:31001 -g &
```

Рис. 2. Команды для установления туннеля между `cluster1` и `cluster0`.

```
[cluster2]$ ssh cluster0 -L31004:node-0-0:31004
-L31005:node-0-1:31005 -g &

[cluster2]$ ssh cluster1 -L31008:node-1-0:31008
-L31009:node-1-1:31009 -g &
```

Рис. 3. Команды для установления туннелей между `cluster2` и `cluster0`, а также между `cluster2` и `cluster1`.

Ключ `-g` указан для возможности использования туннеля с узла кластера. Номера портов назначаются автоматически из диапазона, используемого по умолчанию (начиная с 31000, и увеличиваясь на 4 для каждого соединения). При необходимости номера портов можно переопределить в файле `.netfile`, а также явно указать используемый протокол соединения, например `tcp`, `ssl` или `impi`.

Запуск приложения осуществляется командой `mpirun` с учётом двух дополнительных коммуникационных процессов и указанием переменной окружения `DMPI=pacx`, для выбора DMPI–драйвера. Например, это может выглядеть так, как изображено на рис. 4.

```
[cluster0]$ DMPI=pacx mpirun -np 32 application

[cluster1]$ DMPI=pacx mpirun -np 24 application

[cluster2]$ DMPI=pacx mpirun -np 16 application
```

Рис. 4. Команды запуска распределённого приложения.

4. Модификация кода OpenTS

4.1. Реализация DMPI–драйверов

Как уже отмечалось, поддержка MPI для OpenTS осуществляется с помощью промежуточного интерфейса, называемого DMPI. DMPI представляет собой библиотеку обёрточных функций для некоторого подмножества MPI–вызовов, реализованных в подгружаемом модуле (shared object).

Такое решение позволяет использовать различные реализации MPI без перекомпиляции приложения. С одной стороны это очень полезно при отладке, а с другой, — является необходимым условием для работы в метакластере, так как пока существует требование использовать один исполняемый файл из–за особенностей реализации некоторых механизмов OpenTS.

Необходимый DMPI–модуль определяется в момент запуска по переменным окружения и может быть задан явно с помощью переменных окружения `DMPI` и `DMPI_DEFAULT`. Первая переменная служит для непосредственного указания используемого драйвера в случаях, когда механизм автоматического определения даёт нежелательный результат (например, при неразличимых реализациях MPI), вторая — в случаях невозможности определения нужного драйвера.

Случай с PACX как раз относится к «неразличимым», когда запуск приложения происходит с помощью средств нижележащего MPI и не имеет каких–либо дополнительных особенностей, по которым можно определить режим запуска приложения. В связи с этим обстоятельством необходимо при запуске установить переменную окружения `DMPI` для всех процессов приложения, что может вызвать трудности для некоторых реализаций MPI.

LAM/IMPI и MPICH–G2, в отличие от PACX, устанавливают переменные окружения, по которым их можно отличить, и определение данных реализаций MPI присутствует в DMPI.

Реализации MPI для метакластеров могут также предоставлять информацию о топологии системы. В связи с тем, что отсутствует стандартный способ её получения, каждый случай требует отдельного рассмотрения. Тем не менее, в настоящее время реализованы механизмы определения топологии для перечисленных метакластерных версий MPI и эта информация сделана доступной через интерфейс DMPI.

4.2. Изменения в реализации коммуникационного уровня OpenTS

Несмотря на то, что наличие соответствующих DMPI–драйверов уже позволяет обеспечить запуск приложения на метакластере, потребовалась серьёзная моди-фикация реализации коммуникационного уровня OpenTS. Без такой модифи-кации работа на метакластерных установках в Internet реализована быть не могла, в отличие от некоторых локальных сетей и небольших тестовых конфи-гураций, не обладающих столь значительными неоднородностями

коммуникационного уровня.

Первым шагом на пути модификации транспортного уровня OpenTS, было разделение общей очереди MPI–сообщений на множество очередей для независимого взаимодействия между процессами. Подобное решение отражает внутреннюю структуру используемых реализаций MPI, которые построены на основе TCP–сокетов или отображения памяти удалённого узла. В результате переполнение внутренних буферов в каком–либо направлении не производит нарушения связи в других.

Следующим, и наиболее важным, изменением являлась модификация процедуры отсылки сообщений: вместо непосредственной (in–place) работы с данными для функции `MPI_Isend()` создаётся копия сообщения, чтобы не дожидаться освобождения буферов при необходимости внести изменения. Это обстоятельство сделало также ненужным и средства поиска сообщения в очереди для произведения операции `MPI_Test()` и ожидания завершения операции пересылки.

Несмотря на то, что копирование и создаёт дополнительные накладные расходы, такой способ позволяет обеспечить независимость каналов связи. В дальнейшем планируется рассмотреть возможность реализации копирования при записи (copy–on–write) для пересылаемых данных, не приводящий к нарушению инвариантов на более высоких уровнях структуры OpenTS.

Кроме того, для проведения экспериментов была отключена heartbeat–проверка состояния вычислительных процессов, которая необходима для завершения работы в случае нарушений работы приложения или среды исполнения.

5. Планирование ресурсов

Задача планирования ресурсов является одной из основных при построении системы автоматического динамического распараллеливания. От алгоритма планирования в значительной степени зависит эффективность системы в целом.

Существующий в OpenTS планировщик не подвергался модификации, но, тем не менее, заслуживает подробного рассмотрения. Однако, прежде, чем приступить к описанию собственно планировщика, необходимо сказать несколько слов об общих принципах работы OpenTS.

Основными понятиями OpenTS являются «T–функция» и «неготовое значение». С их помощью обеспечивается параллельное выполнение приложения. При вызове T–функции, которая оформляется в коде специальным образом, производится постановка её в очередь вместо непосредственного исполнения, а всем выходным результатам приписывается значение «не готов». Над неготовыми значениями определены операции присваивания и ожидания результата. В частности, подобные значения можно передавать в качестве аргументов другим T–функциям или возвращать в качестве результата. Основное требование, предъявляемое к T–функциям —

отсутствие побочных эффектов, что гарантирует корректность результата вне зависимости от порядка проведения вычислений и позволяет производить вычисления параллельно.

Все вычисляемые T–функции приложения разделены на два класса: уже начавшие выполнение в данном процессе и ещё не начавшие (так называемые «пренатальные» T–функции). В связи с тем, что миграция T–функций, начавших исполнение, пока невозможна из–за трудностей перенесения стека, они хранятся в двух независимых очередях. Пока есть возможность продолжать выполнение уже начатых задач, функции из «пренатальной» очереди для локального исполнения не извлекаются.

Управление очередью «пренатальных» T–функций происходит так, что при локальном исполнении реализуется обход графа задачи «в глубину», а именно: T–функции извлекаются из начала очереди (в терминах функционального программирования [2] такой порядок редукции графа называется аппликативным), то есть, фактически, это стек выполнения задачи. При отсылке T–функции для внешнего исполнения функции извлекаются из «хвоста» очереди, реализуя обход «в ширину» (нормальный порядок редукции). Подобная схема позволяет избегать переполнения памяти и обеспечивает возможность эффективной реализации рекурсивных задач наподобие рассматриваемой.

Каждый вычислительный процесс содержит информацию о состоянии других процессов, выраженную в виде одного из трёх состояний: «перегружен», «занят» и «свободен». Если нет T–функций, способных продолжать выполнение и при выборе очередной T–функции для выполнения обнаруживается, что доступно более одной T–функции в «пренатальном» состоянии, то определяется свободный вычислительный процесс с наибольшей производительностью и T–функция посылается ему. При этом процесс помечается как занятый. В случае, когда не удаётся найти свободный процесс, публикуется перегруженность узла. При одном «пренатальном» процессе в очереди публикуется занятость узла, а при пустой очереди — то, что он свободен.

Операция оповещения о состоянии ресурса осуществляется рассылкой сообщения всем процессам и использует кэш, чтобы сообщать только об изменениях состояния.

Независимость очередей сообщений позволяет неявно использовать информацию о пропускных способностях каналов. При этом скорость обмена состоянием процессов зависит от коммуникационных возможностей среды, и, как следствие, чем лучше характеристики канала, тем выше вероятность передачи по нему функции, и, соответственно, данных. Таким образом, даже без дополнительных средств удаётся приспособиться под возможности оборудования.

Заметим, что подобное решение обладает ограниченной масштабируемостью и должно быть пересмотрено при дальнейшем увеличении числа вычислительных процессов. Эти и другие направления дальнейших исследований будут рассмотрены далее в разделе 9.

6. Используемое аппаратное и программное обеспечение

В качестве составляющих компонент метакластера рассматривались три вычислительных кластерных системы на основе внутренней высокоскоростной сети SCI (Scalable Coherent Interface) под управлением ОС Linux, расположенных в Исследовательском центре многопроцессорных систем Института программных систем РАН (ИЦМС ИПС РАН, г. Переславль–Залесский), в НИИ Механики МГУ им. М. В. Ломоносова (г. Москва) и в ОИПИ НАН Республики Беларусь (г. Минск).

Характеристики этих систем приведены в таблице 1.

	НИИМ МГУ	ИЦМС ИПС РАН	ОИПИ НАН РБ
Nodes	8	16	64
CPUs/node	2		
CPU model	Athlon MP 1800+		Pentium 4 Xeon
CPU, MHz	1533		2800
Memory, GiB	1,0		2,0
MPI	SCI–MPICH	Scali	
SCI	D335/D334	D335	D336
SCI Topology	4×2	4×4	4×4×4

Таб. 1. Характеристики используемых кластеров

Для проведения экспериментов PACX был скомпилирован с SCI–MPICH на кластере НИИ механики МГУ и со Scali MPI на кластерах «СКИФ» ИЦМС ИПС РАН и «К500» ОИПИ НАН РБ. Потребовались незначительные изменения кода PACX, касающиеся в основном скрипта компиляции `pacxcc`.

В тестировании принимали участие реализация EP из пакета NPB–2.3 и EP для модифицированного варианта OpenTS. Задачи компилировалась с оптимизацией под процессор Athlon MP.

Реализация EP для OpenTS основана на рекурсии, так что вызовы функций образуют двоичное дерево. Листья этого дерева производят основную часть вычислений, а промежуточные вершины только совершают вызовы и суммируют результат. Объём вычислений и глубина рекурсии задаются в виде параметров командной строки.

У теста NPB параметры задаются при компиляции и включают число узлов и так называемый «класс» задачи. Используемые тесты классов A и C соответствуют OpenTS EP параметрами размера задачи 28 и 32.

7. Результаты экспериментов

Реализация NPB EP мало зависит от коммуникаций, поэтому время её выполнения можно считать линейно зависящим от размера задачи. Серия запусков продемонстрировала стабильность временных характеристик теста, и полученное время выполнения было принято за основу для расчёта утилизации.

Для запусков с участием вычислительного кластера «К500» утилизация не считалась, так как во-первых не проводилось контрольных запусков NPB EP, а во-вторых, отсутствовала возможность выполнения достаточного для оценок количества тестов, таким образом, в случае «К500» установлены только факты запуска.

Результаты тестовых запусков представлены в таблице 2, где используются следующие обозначения.

- Test — реализация теста: NPB для NPB–2.3 EP, OTS для OpenTS EP.
- Sys — состав вычислительной системы: I — кластер НИИ механики МГУ, S — кластер «СКИФ» ИЦМС ИПС РАН, K — кластер «К500» ОИПИ НАН РБ. Вычислительные кластеры перечислены в порядке назначения MPI–рангов.
- Topology — топология кластера. Слагаемые обозначают количество вычислительных процессов на задействованных кластерах. Порядок соответствует перечислению в поле Sys. Индекс «$_{/2}$» обозначает выполнение по два процесса на каждом из задействованных узлов данного кластера.
- N — количество вычислительных процессов.
- P — общее число процессов.
- S — размер задачи, объём вычислений составляет $Q(2^{\Sigma})$.
- D — глубина рекурсии, объём вычислений гранулы параллелизма составляет $Q(2^{\Sigma-\Delta})$, количество гранул 2^{D}.
- Time — время выполнения в секундах.
- Производные величины:

ITime — идеальное время выполнения в секундах, рассчитанное на основе времени выполнения NPB EP класса C (S=32) на одном процессоре кластера НИИ механики МГУ по формуле $\frac{1749{,}23 \times 2^{S-2}}{N}$. В расчёте идеального времени выполнения участвует N, а не P из–за особенностей коммуникационной среды — для запуска NPB EP под PACX также будут доступны для вычислительной работы N из P процессов.

Util — утилизация вычислительной мощности системы, $\mathrm{Util} = \frac{\mathrm{ITime}}{\mathrm{Time}} \times 100\%$.

Test	Sys	*N*	*P*	Topology	*S*	*D*	Time(s)	ITime(s)	Util(%)
NPB	I	1	1	1	28	—	109,80	109,32	99,57
NPB	I	16	16	$16_{/2}$	32	—	109,41	109,32	99,92
NPB	I	1	1	1	32	—	1749,23	1749,23	100,00
OTS	I	1	1	1	28	0	115,033	109,32	95,04
OTS	I	1	1	1	28	20	157,134	109,32	69,57
OTS	I	1	1	1	32	0	1832,044	1749,23	95,47
OTS	I	8	8	$8_{/2}$	32	10	233,533	218,654	93,63
OTS	I	8	8	$8_{/2}$	32	18	232,400	218,654	94,09
OTS	S	32	32	$32_{/2}$	34	14	283,716	218,654	77,07
OTS	S	32	32	$32_{/2}$	36	16	992,689	874,615	88,11
OTS	S	32	32	$32_{/2}$	38	18	3825,094	3498,46	91,46
OTS	S	32	32	$32_{/2}$	40	20	15159,644	13993,84	92,31
OTS	K	16	16	16	32	10	140,972	—	—
OTS	K, S	44	48	$16+32_{/2}$	36	18	6443,206	—	—
OTS	S, I	40	44	$32_{/2}+12_{/2}$	0	0	49,563	—	—
OTS	S, I	40	44	$32_{/2}+12_{/2}$	34	14	332,164	205,576	61,89
OTS	S, I	40	44	$32_{/2}+12_{/2}$	36	16	882,184	822,306	93,21
OTS	S, I	40	44	$32_{/2}+12_{/2}$	36	20	877,080	822,306	93,75
OTS	S, I	40	44	$32_{/2}+12_{/2}$	38	14	3289,223	2798,768	85,09
OTS	S, I	40	44	$32_{/2}+12_{/2}$	38	18	3137,913	2798,768	89,19
OTS	S, I	40	44	$32_{/2}+12_{/2}$	38	22	3116,891	2798,768	89,79
OTS	S, I	40	44	$32_{/2}+12_{/2}$	38	24	3221,088	2798,768	86,89
OTS	S, I	40	44	$32_{/2}+12_{/2}$	38	26	3353,231	2798,768	83,46
OTS	S, I	40	44	$32_{/2}+12_{/2}$	40	20	12132,440	11195,072	92,27

Таб. 2. Результаты экспериментов

8. Анализ результатов

Прежде всего, следует отметить, что эксперименты проводились без использования каналов с гарантированным качеством обслуживания, а, следовательно, результаты повторных запусков приложения могут иметь значительные отклонения от приведённых в таблице. Также, в связи

с необходимостью увеличения вычислительной сложности задачи для снижения влияния времени запуска приложения, не представляется возможным проведение достаточного числа запусков, чтобы стало возможным применение средств статистического анализа.

Тем не менее, полученные результаты позволяют обнаружить параметры, существенно влияющие на время выполнения приложения и приблизительно оценить их оптимальные значения.

8.1. Внешние по отношению к приложению факторы

В связи с ростом числа вычислительных узлов, возрастает время запуска приложения, а также возникает необходимость в обеспечении синхронизации распределённого запуска. Результатом является низкая эффективность использования ресурсов при непродолжительном выполнении.

Также следует отметить потерю эффективности, возникающую из–за необходимости создания коммуникационных процессов.

Перечисленные свойства среды исполнения делают оправданным применение метакластеров в задачах, где действительно требуется использование значительного количества вычислительных ресурсов. При этом отсутствует возможность разделения задачи на независимые части, не требующие обмена данными в процессе выполнения.

Здесь же следует обратить внимание на необходимость непрерывной работы TCP–соединений, что может оказаться проблематичным, как в силу естественных причин, так и из–за возможной активности злоумышленников.

8.2. Влияние характеристик приложения

Наиболее интересным является исследование времени выполнения задания в зависимости от значения параметра глубины рекурсии *D*. С одной стороны, меньшее значение данного параметра сокращает расходы на порождение новых функций, что может вносить ощутимый вклад при большом размере задачи. С другой стороны, увеличение *D* приводит к уменьшению размера гранулы параллелизма и позволяет сократить задержки коммуникационного уровня при обработке сообщений.

Как следствие, для эффективного выполнения задания в целом, подбор данного параметра очень важен.

9. Направления дальнейших исследований

Представленные в настоящей работе результаты являются лишь первыми шагами в относительно новой, очень динамичной, и пока ещё мало исследованной области. Однако, они создают экспериментальную и методологическую базу для дальнейшего поиска.

К числу основных направлений для дальнейших исследований можно отнести расширение класса успешно–решаемых задач и увеличение масштабируемости транспортного уровня.

Представленные эксперименты не использовали информацию о топологии вычислительной системы, но она может оказаться необходимой в задачах с более сложным алгоритмом работы или при значительном увеличении числа узлов, когда организация попарных соединений между всеми узлами кластера станет создавать проблемы из–за немасштабируемости подобного подхода.

В качестве увеличения масштабируемости могут быть использованы подходы на основе рассмотренного в [14], но данная проблема требует более детального исследования с проведением экспериментов на реальных вычислительных системах.

10. Заключение

Представленные результаты являются первым практическим исследованием возможностей системы автоматического динамического распараллеливания OpenTS на реальных метакластерных установках в Internet. Несмотря на ряд пока ещё нерешённых проблем и нереализованных возможностей, система уже смогла продемонстрировать высокие показатели утилизации вычислительной мощности на задаче из достаточно широкого класса, где применение систем автоматического динамического распараллеливания является оправданным, что, наряду с другими достижениями OpenTS, свидетельствует об успешном преодолении начального этапа разработки и вступлении в стадию развития и совершенствования.

Автор выражает благодарность за участие в постановке задачи, внимание к работе и обсуждение результатов своему научному руководителю профессору В. А. Васенину и старшему научному сотруднику НИИ механики МГУ В. А. Роганову.

Ссылки

1. Динамическое распараллеливание программ на базе параллельной редукции графов. Архитектура программного обеспечения новой версии Т–системы / С.М.Абрамов, В.А.Васенин, Е.Е.Мамчиц и др. // Высокопроизводительные вычисления и их приложения: Труды Всероссийской научной конференции. "— М.: Изд–во МГУ, 2000. "— г. Черноголовка, 30 октября–2 ноября. "— С. 261–265.
2. А.Филд, П.Харрисон. Функциональное программирование: Пер. с англ. "— М.: «Мир», 1993. "— С. 637.
3. В.А.Васенин, В.А.Роганов. GRACE: распределённые вычисления в Internet // Открытые Системы. СУБД. "— 2001. "— 5–6.
4. А.В.Инюхин. К поддержке метакластерных структур в среде Internet // Технологии высокопроизводительных информационно–вычислительных систем / Под ред. проф. В. А. Васенина. "— Изд–во «Университет города Переславля», 2003. "— С. 111–118.

5. Burns G., Daoud R., Vaigl J. LAM: An Open Cluster Environment for MPI // Proceedings of Supercomputing Symposium. "— 1994. "— Pp. 379–386. http://www.lam-mpi.org/download/files/lam-papers.tar.gz.
6. The Condor project. http://www.cs.wisc.edu/condor/.
7. The Globus project. http://www.globus.org/.
8. IMPI homepage. http://impi.nist.gov/IMPI/.
9. The interoperable message passing interface (IMPI) extensions to LAM/MPI // Proceedings, MPIDC'2000. "— 2000. "— mar.
10. MPICH homepage. http://www-unix.mcs.anl.gov/mpi/mpich/indexold.html.
11. NAS Parallel Benchmarks. http://www.nas.nasa.gov/Software/NPB/.
12. PACX homepage. http://www.hlrs.de/organization/pds/projects/pacx-mpi/.
13. S.Abramov, A.Adamovitch, M.Kovalenko. T–system: programming environment providing automatic dynamic parallelizing on IP–network of UNIX–computers. "— Moscow: 1997. "— Sept. http://www.botik.ru/ abram/T-System-970915/report.html.
14. Supressing roughness of virtual times in parallel discrete–event simulations / G.Korniss, M.A.Novotny, H.Guclu et al. // Science. "— 2003. "— Vol. 299. "— Pp. 668–669.