Программная архитектура системы передачи интенсивного потока данных в распределенных системах*

В.А. Щапов[1,2]

Институт механики сплошных сред УрО РАН[1],
Пермский национальный исследовательский политехнический университет[2]

В статье рассмотрена программная архитектура системы передачи интенсивного потока данных в распределенных системах. Предложенная архитектура реализует разработанную в ИМСС УрО РАН модель потоковой обработки данных от экспериментальной установки PIV (Particle Image Velocimetry) на удаленных высокопроизводительных вычислительных системах с передачей данных по скоростной оптической сети большой протяженности. Описаны класс прикладных задач, для которых можно использовать систему, и схема взаимодействия распределенных компонент программного обеспечения (ПО). Приводятся результаты измерений, иллюстрирующие эффективность разработанных алгоритмов диспетчеризации параллельных потоков данных и протокола взаимодействия оконечных систем.

1. Введение

Современные экспериментальные установки генерируют большие объемы данных, нуждающихся в обработке в реальном времени. Одной из таких задач является обработка экспериментальных данных, получаемых по методу PIV (Particle Image Velocimetry) – оптическому методу измерения полей скорости жидкости или газа в выбранном сечении потока. Метод основан на обработке пар фотографий трассеров — мелких частиц, взвешенных в потоке, в моменты, когда они подсвечиваются импульсным лазером, создающим тонкий световой нож. Скорость потока определяется расчетом перемещения трассеров за время между вспышками лазера. Интенсивность порождаемого потока данных зависит от числа, разрешения и частоты работы камер и может достигать нескольких гигабит в секунду [1]. Обработка таких потоков данных с использованием только локального компьютера экспериментальной установки (ЭУ) чрезвычайно затруднена. В то же время развитие математического аппарата и появление новых высокоточных алгоритмов расчетов увеличивают требования к необходимой вычислительной мощности. Поэтому перенос вычислений на удаленные суперкомпьютеры достаточной производительности позволит применять новые высокоточные алгоритмы, обрабатывать данные в реальном времени, уменьшить объемы сохраняемых на ЭУ данных. В ИМСС УрО РАН данное направление развивается в рамках проекта «Распределенный PIV» [2] с использованием высокоскоростной оптической магистрали, создаваемой в рамках проекта «Инициатива GIGA UrB RAS» [3].

Разработанная программная архитектура использует лямбда грид парадигму распределенных вычислений [4], в которой используется параллелизм потоков данных в скоростных оптических сетях со спектральным уплотнением каналов. Достижение эффективной диспетчеризация параллельных потоков между сопрягаемыми системами является основной целью представленной работы. Измерения проводились по выделенному L2 каналу связи, объединяющий в единую подсеть ЭУ (Пермь, ИМСС УрО РАН), суперкомпьютеры «Уран» и «UM64» и систему хранения данных (СХД) EMC Celerra NS-480 (Екатеринбург, ИММ УрО РАН). В настоящий момент доступная пропускная способность выделенного канала связи составляет 1 Гбит/с.

2. Задачи, допускающие параллельную обработку потока данных

Анализ исходной задачи потоковой обработки исходных данных эксперимента PIV показал, что каждое измерение можно обрабатывать независимо от других [5]. Исходя из этого, бы-

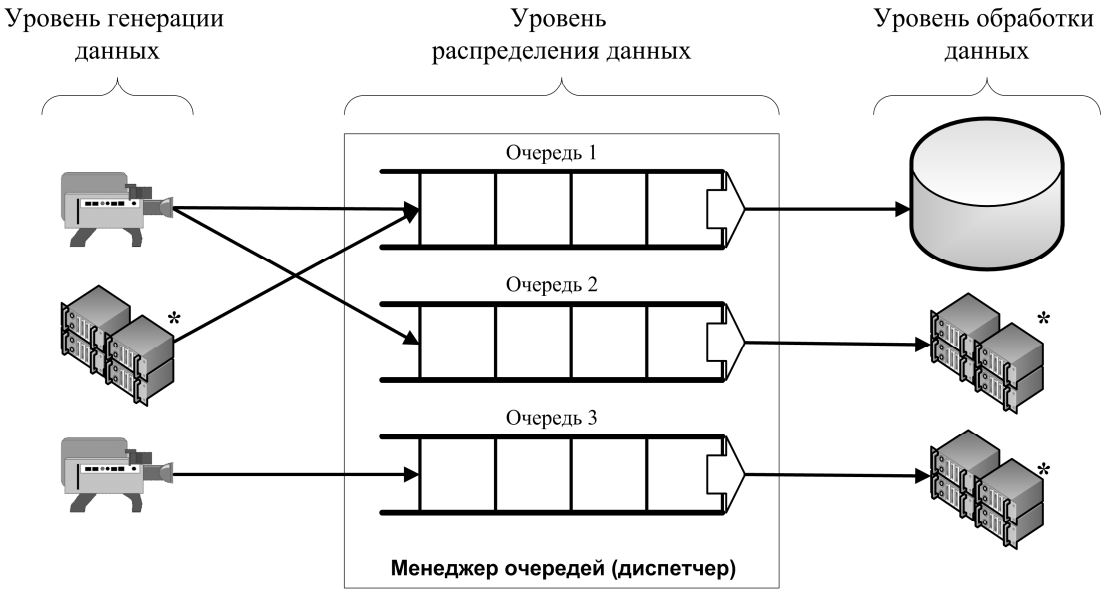
* Исследование проводится при поддержке РФФИ (грант № 11-07-96001-р_урал_а).

ло предложено отказаться от однозначного отображения измерений на вычислительные узлы и применить концепцию очередей для обработки данных [6].

Обобщение концепции очередей позволило сформулировать границы применимости подхода для расчетных алгоритмов. Параллельная потоковая обработка исходных данных возможна только тогда, когда расчетный алгоритм позволяет выделить во множестве исходных данных независимые подмножества, допускающие независимую обработку.

3. Трехуровневая программная архитектура

Возможность независимой обработки подмножеств исходных данных позволило применить концепцию очередей для решения задачи передачи и диспетчеризации потока данных (рис. 1).



*) Одно устройство может находиться и на уровне генерации данных, и на уровне обработки данных

Рис. 1. Трехуровневая программная архитектура

Концепция очередей приводит к разделению системы на три уровня:

1. уровень генерации данных;
2. уровень распределения данных;
3. уровень обработки данных.

Распределение очереди проводится по принципу First In First Out (FIFO). При этом передача данных между уровнями реализуется с использованием протокола, получившего название «Протокол PIV», построенного на идее модели RPC (рис. 2). В этом случае сервером является менеджер очередей, реализующий уровень распределения данных. Клиентами является ПО на уровнях генерации и обработки данных.

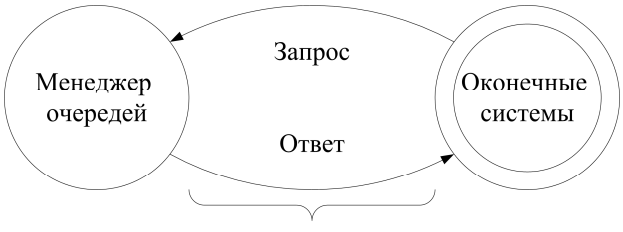


Рис. 2. Схема взаимодействия компонент

Выделение задачи диспетчеризации в отдельный слой, который может располагаться, как на ЭУ, так и на отдельном сервере, позволило отказаться от межузлового обмена данными на стороне суперкомпьютера, что дает следующие преимущества:

- автоматическая балансировка нагрузки по вычислительным узлам — более быстрые узлы будут чаще посылать запросы новых данных и, таким образом, будут получать больше данных для обработки;
- возможность изменять число используемых вычислительных узлов во время проведения эксперимента;
- возможность использовать вычислительную мощность нескольких суперкомпьютеров;
- минимизация потери данных в случае выхода из строя одного или нескольких вычислительных узлов (в худшем случае потеряется только текущее обрабатываемое измерение сбойного узла). В случае если потеря данных недопустима, то на время расчета измерения необходимо сохранять в памяти сервера и, в случае выявления сбоя, повторно добавлять эти блоки в очередь готовых к обработке измерений.

Необходимо отметить, что предлагаемый подход не ограничивает нас в выборе транспорта для передачи данных. Текущая реализация использует для передачи данных протокол PIV. Однако, возможно и использование общего СХД, подключенного и к ЭУ, и к супервычислителю. В этом случае задачей диспетчера будет распределение по вычислительным узлам не самих данных, а путей к файлам с данными на СХД, при этом вычислительные узлы будут считывать данные непосредственно из файлов на СХД. Это позволит анализировать и сравнивать поведение, как классической схемы с общим дисковым пространством, так и предлагаемой схемы без промежуточного хранения данных на дисковых массивах.

Так как технология позволяет вынести менеджер очередей на отдельный сервер, это может позволить снизить нагрузку на ЭУ благодаря переносу задачи по обслуживанию большого числа параллельных TCP-соединений на отдельный сервер, при этом ЭУ будет передавать данные менеджеру очередей через небольшое число TCP-соединений. Так как менеджер очередей будет располагаться недалеко от ЭУ, то даже небольшое число TCP-соединений будут полностью утилизировать канал связи между ЭУ и менеджером очередей.

На рис. 3 показано применение предложенной архитектуры в проекте «Распределенный PIV».

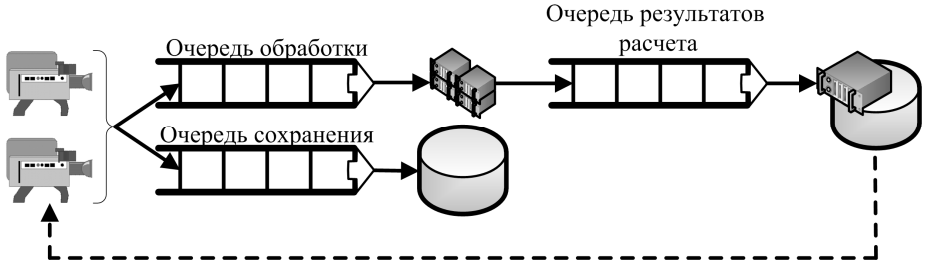


Рис. 3. Применение трехуровневой программной архитектуры в проекте «Распределенный PIV»

Данные с камер ЭУ загружаются в одну или две очереди: очередь обработки и очередь сохранения. Данные из очереди сохранения забираются ПО, которое запускается на хранилище данных, что позволяет сохранить исходные данные эксперимента в случае невозможности их сохранения на ЭУ по техническим причинам. Данные из очереди обработки передаются на вычислительные узлы суперкомпьютеров, результаты обработки помещаются в очередь результатов расчетов, откуда их получает ПО, работающее на ЭУ. Полученные ЭУ результаты расчета могут использоваться для управления экспериментом. При необходимости сохранения результатов расчетов на внешнем хранилище они также могут помещаться в две или более очереди, из одной из которых данные будут забираться ПО хранилища данных.

3.1 Протокол PIV

Протокол PIV является протоколом прикладного уровня, реализующим идею модели RPC. Протокол работает по схеме запрос-ответ и может использовать в качестве протокола транс-

портного уровня любой надежный потоковый протокол передачи данных. Текущая реализация протокола PIV поддерживает транспортные протоколы TCP и UDT [7]. Положение протокола PIV в стеке сетевых протоколов показано на рис. 4 [8].

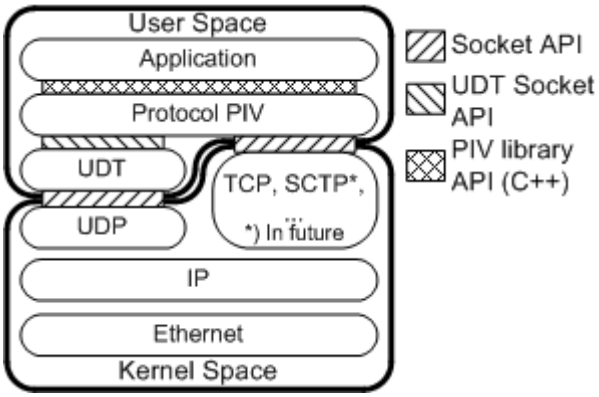


Рис. 4. Положение протокола PIV в стеке сетевых протоколов

Протокол UDT – это, основанный на UDP, протокол передачи данных для высокоскоростных сетей. Он был разработан в Университете штата Иллинойс в Чикаго. Функциональные возможности протокола UDT аналогичны протоколу TCP. UDT является дуплексным протоколом передачи потока данных с предварительной установкой соединения. Особенностью протокола UDT является оригинальная архитектура и реализация, а также оригинальный алгоритм управления перегрузкой. При этом протокол UDT позволяет программисту реализовать и использовать свой алгоритм управления перегрузкой.

Протокол PIV рассчитан на передачу нескольких именованных блоков бинарных данных. В одном пакете протокола PIV можно передать от нуля до 65535 блоков, каждый из которых может иметь размер до 4 Гбайт, при этом сохранение порядка следования блоков не гарантируется. Длина имени блока ограничена 255 байтами. На рис. 5 приведен формат пакета протокола PIV. Поля заголовка пакета протокола кодируются в сетевом порядке байт.

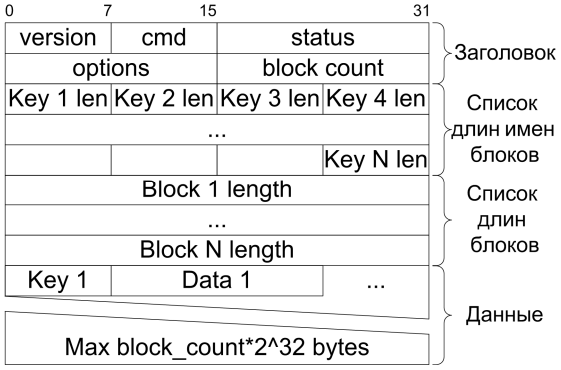


Рис. 5. Формат пакета протокола PIV

Протокол поддерживает версионирование с использованием поля `version`, что позволяет реализовать в рамках одной версии ПО поддержку нескольких версий протоколов.

Поле `cmd` содержит один из следующих кодов команд:

- `SciMQ_PROTOCOL_CMD_RESPONSE` – пакет ответа сервера на запрос клиента;
- `SciMQ_PROTOCOL_CMD_PING` – запрос, отправляемый клиентом для проверки активности канала связи;
- `SciMQ_PROTOCOL_CMD_GET` – запрос данных у сервера;
- `SciMQ_PROTOCOL_CMD_POST` – отправка данных на сервер;
- `SciMQ_PROTOCOL_CMD_CONFIRM` – команда подтверждения приема данных при использовании надежных очередей;
- `SciMQ_PROTOCOL_CMD_CREATE_QUEUE` – команда создания очереди;
- `SciMQ_PROTOCOL_CMD_DELETE_QUEUE` – команда удаления очереди;

- `SciMQ_PROTOCOL_CMD_GETINFO_QUEUE` – команда получения информации об очереди;
- `SciMQ_PROTOCOL_CMD_MODIFY_QUEUE` – команда изменения очереди.

Поле `status` предназначено для передачи клиенту статуса ответа от сервера. В зависимости от значения статуса можно судить о выполнении или невыполнении запроса клиента.

Поле `options` предназначено для кодирования дополнительных опций.

Поле `block_count` содержит число передаваемых блоков данных.

Далее в пакете располагаются список длин имен блоков и список длин блоков, после чего передаются пары, состоящие из имени блока и данных блока.

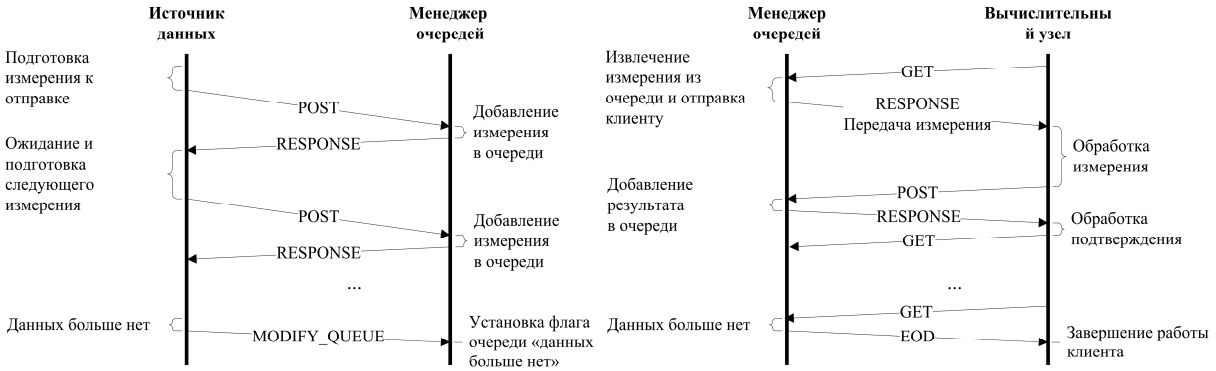


Рис. 6. Временная диаграмма работы протокола PIV

На рис. 6 приведена временная диаграмма работы протокола PIV в трехуровневой программной архитектуре. Левая часть диаграммы показывает взаимодействие уровня генерации данных, а правая часть – уровня обработки данных с менеджером очередей. Для наглядности эти процессы разделены, однако существенным является то обстоятельство, что передача по протяженной линии связи занимает значительное время и процессы передачи необходимо выполнять одновременно при наличии данных.

3.2 Уровень генерации данных

Уровень генерации данных реализует загрузку исходных данных из источника и отправку их на уровень распределения данных, используя для взаимодействия с сервером очередей специальную клиентскую библиотеку.

Существуют различные стратегии загрузки исходных данных. В случае использования, например, ПО ActualFlow данные могут загружаться из файлов в файловой системе ЭУ. В более сложных случаях, когда требуется обрабатывать данные от большого числа источников, возможна передача данных непосредственно с датчиков, например с камер ЭУ, обладающих Ethernet-интерфейсом и позволяющих реализовать логику протокола PIV.

3.3 Уровень распределения данных

Уровень распределения данных решает задачу получения блоков данных от уровня сбора данных и их передачу на уровень обработки данных. В связи с тем, что уровень обработки данных состоит из множества вычислительных процессов, запущенных на узлах суперкомпьютера, на диспетчер ложится задача по распределению очередей данных на множество узлов суперкомпьютера.

В настоящий момент реализован подход, когда, при наличии данных в очереди, они будут отдаваться сервером на все поступающие запросы, при наличии доступных ресурсов на сервере. Однако, возможно, что с точки зрения загрузки сети и отзывчивости системы, данный подход не будет являться оптимальным. Исследование поведения системы и выбор оптимальных стратегий распределения очереди в условиях большого числа вычислительных узлов (более 512) будут предметом дальнейших исследований. Предполагается определить такую конфигу-

рацию параметров, при которой будет соблюдаться баланс между числом активных параллельных соединений, уровнем загрузки сети и временем получения результата расчета после начала его отправки (отзывчивость системы).

3.4 Уровень обработки данных

На этом уровне происходит обработка данных. Это может быть расчет исходных данных с применением каких-либо алгоритмов, сохранение данных в высокопроизводительных хранилищах или на большом количестве локальных дисков, передача данных из очередей в сторонние системы и т.д.

Как и уровень генерации, уровень обработки данных взаимодействует с сервером очередей уровня распределения с помощью специальной клиентской библиотеки.

4. Программная реализация

Предложенная программная архитектура была реализована в виде комплекса ПО для всех уровней архитектуры.

Были разработаны три компоненты общего назначения:

- сервер очередей,
- клиентская библиотека,
- управляющее ПО.

И два специализированных компонента для PIV-расчетов:

- клиент загрузки исходных данных и сохранения результатов расчетов,
- клиент, упрощающий написание расчетных алгоритмов для суперкомпьютера.

Все приложения и библиотеки разработаны на языке программирования C++ с использованием библиотек Boost[1]. Использование Boost позволило ускорить разработку с выполнением требований по кроссплатформенности ПО, а выбор языка программирования C++ и использование оптимизирующих компиляторов GCC и Intel C++ позволил разрабатывать высокопроизводительный код.

Разработанный комплекс ПО поддерживает работу под управлением операционных систем Red Hat Enterprise Linux 5 и новее, SUSE 11 и новее, и Windows 7 и новее. Поддерживаются компиляторы GCC 4.1.2 и новее, Intel C++ 11 и новее, и Microsoft Visual C++ 2010 (10.0).

4.1 Сервер очередей

Сервер очередей реализует функциональность менеджера очередей (диспетчера). Архитектурно он состоит из четырех компонент:

- управляющий модуль,
- модуль очередей,
- модуль фоновых процессов,
- модуль сетевого взаимодействия.

Все модули, за исключением менеджера очередей, работают по событийно-ориентированному принципу с использованием технологий Boost.ASIO.

Управляющий модуль отвечает за начальную инициализацию сервера, загрузку параметров конфигурации, запуск и остановку остальных модулей, обработку сигналов операционной системы, таких как сигналы немедленного и плавного завершения и сигнал для переоткрытия файлов журналов. Цикл обработки сообщений реализуется объектом `boost::asio::io_service`, метод `run()` которого выполняется непосредственно в главном потоке приложения.

Модуль очередей управляет хранением очередей в памяти сервера. Каждая очередь имеет свое уникальное имя и может быть с гарантией обработки или без гарантии обработки. Если очередь без гарантии обработки, то элемент удаляется из очереди сразу же после отправки кли-

[1] Boost C++ Libraries: http://www.boost.org/

енту. Если очередь с гарантией обработки, то, после передачи клиенту, элемент помещается во временный список, из которого удаляется после получения подтверждения обработки или перемещается в основную очередь, если в течение задаваемого пользователем таймаута подтверждение получено не было. Однако использование таких очередей повышает требования к доступной оперативной памяти, так как данные приходится хранить дольше.

Для распределения памяти модуль очередей использует два менеджера памяти, которые позволяют отдельно настроить ограничение на максимальное потребление памяти для структур самой очереди и для данных, помещаемых в очередь.

Модуль фоновых процессов предназначен для выполнения низкоприоритетных отложенных задач и состоит из одного объекта-диспетчера событий `boost::asio::io_service`, методы `run()` которого выполняются в одном или более независимых потоков. Это позволяет запускать в рамках одного диспетчера событий длительные задачи, так как, пока выполняется одна задача, остальные задачи могут обрабатываться оставшимися потоками. Модуль фоновых процессов используется для отслеживания таймаутов в очередях с гарантией обработки.

Модуль сетевого взаимодействия отвечает за реализацию взаимодействия с клиентами по сети, то есть непосредственно реализует функциональность протокола PIV. Он представляет собой пул потоков, в каждом из которых есть свой объект-диспетчер событий `boost::asio::io_service`, метод `run()` которого выполняется в этом же потоке.

Для возможности обработки нескольких клиентских соединений одним потоком применяются неблокирующие сокеты, асинхронные операции и зависимые от операционной системы технологии мультиплексирования, которые реализуются внутри библиотеки Boost.ASIO. Каждому соединению с клиентом в момент его создания назначается один из объектов `io_service`, который будет обслуживать все асинхронные операции в этом соединении. Выбор такой архитектуры вызван тем, что большинство операционных систем не поддерживают параллельное ожидание событий на одном наборе сокетов из нескольких потоков. Из-за этого клиентские сокеты принудительно распределяются по нескольким потокам, что позволяет снизить накладные расходы на блокировки, которые в условиях небольшого времени работы обработчиков событий становятся значительными.

4.2 Клиентская библиотека

Клиентская библиотека предназначена для упрощения написания прикладных приложений, так как снимает с программиста необходимость написания собственной реализации протокола PIV. Клиентская библиотека `libscimqclient` предоставляет прикладному программисту синхронный API для взаимодействия с сервером очередей и скрывает все нюансы реализации протокола PIV и работы с сетью. API клиентской библиотеки показан на рис. 7.

```
namespace SciMQ { namespace Client {
class LIBSCIMQCLIENT_API ProtocolException: public std::exception
{
public:
  ProtocolException(int status) throw();
  virtual ~ProtocolException() throw();
  scimq_protocol_status_t status() const throw();
  virtual const char* what() const throw();
};

class LIBSCIMQCLIENT_API Connection: private boost::noncopyable
{
public:
  Connection();
  Connection(
    const SciMQ::Network::EndpointType& endpoint,
    const SciMQ::Network::EndpointOptionType& endpoint_options =
SciMQ::Network::EndpointOptionType()
  );
  ~Connection();
```

```
  void connect(
    const SciMQ::Network::EndpointType& endpoint,
    const SciMQ::Network::EndpointOptionType& endpoint_options =
SciMQ::Network::EndpointOptionType()
  );
  void close();
  // Запрос данных для обработки.
  // Если возвращенный указатель равен NULL, то данных больше нет.
  SciMQ::DataQueue::MessageSharedPtr get_data(
    const SciMQ::DataQueue::MessageKeyType& queue,
    bool& is_data_ended
  );
  // Отправка результатов на сервер.
  void post_data(
    const SciMQ::DataQueue::MessageKeyListType& queue_list,
    SciMQ::DataQueue::MessageSharedPtr& message,
    bool is_push = false,
    const SciMQ::DataQueue::MessageKeyType& confirm_queue =
SciMQ::DataQueue::MessageKeyType(),
    const SciMQ::DataQueue::MessageUuidType& confirm_uuid =
SciMQ::DataQueue::make_message_nil_uuid()
  );
  // Отправка подтверждения на сервер.
  void post_confirm(
      const SciMQ::DataQueue::MessageKeyType& confirm_queue,
      const SciMQ::DataQueue::MessageUuidType& confirm_uuid,
      bool is_push = false
  );
  // Управление очередями
  void create_queue(
    const SciMQ::DataQueue::MessageKeyType& queue,
    bool is_reliable   = false,
    bool is_persistent = false,
    const boost::chrono::seconds& reliable_timeout =
SciMQ::DataQueue::MESSAGE_RELIABLE_TIMEOUT
  );
  void delete_queue(
    const SciMQ::DataQueue::MessageKeyType& queue
  );
  bool get_queue_info(
    const SciMQ::DataQueue::MessageKeyType& queue,
    bool *is_reliable = NULL,
    bool *is_persistent = NULL,
    bool *is_eod = NULL,
    boost::chrono::seconds *reliable_timeout = NULL
  );
  bool exist_queue(const SciMQ::DataQueue::MessageKeyType& queue);
  void set_queue_eod(const SciMQ::DataQueue::MessageKeyType& queue, bool
eod);
  bool get_queue_eod(const SciMQ::DataQueue::MessageKeyType& queue);
};
} }
```

Рис. 7. API клиентской библиотеки libscimqclient

4.3 Управляющее программное обеспечение

Управляющее ПО предназначено для управления очередями. Дополнительно поддерживается возможность генерации тестового потока данных, загрузка данных с опциональной под-

держкой сохранения данных в файлах, а также загрузка данных из одной очереди и помещение их в другую очередь.

4.4 Программное обеспечение для PIV-расчетов

Данный компонент разработанного ПО состоит из двух частей:

- клиент загрузки исходных данных и сохранения результатов расчетов,
- клиент, упрощающий написание расчетных алгоритмов для суперкомпьютера.

Приложение ввода-вывода предназначено для загрузки исходных данных из формата хранения ActualFlow и для сохранения получаемых результатов в файлах на дисках ЭУ. Поддерживаются два режима загрузки исходных файлов. В первом случае, будут рекурсивно прочитаны все заданные каталоги, и обнаруженные файлы данных будут переданы на сервер, после чего приложение завершается. Во втором случае, помимо обнаружения всех существующих данных в заданном каталоге, будет включен мониторинг появления новых файлов данных и каталогов, которые будут отправляться на сервер по мере их появления. В этом случае, для завершения работы требуется подать специальную команду в консольном окне приложения. Для завершения наблюдения с продолжением передачи уже обнаруженных данных на сервер нужно нажать комбинацию клавиш `Ctrl+C` один раз. Для завершения работы без передачи уже обнаруженных данных на сервер необходимо нажать комбинацию клавиш `Ctrl+C` два раза.

Расчетный клиент предназначен для упрощения написания расчетных приложений на стороне суперкомпьютера. Он берет на себя функции загрузки параметров из конфигурационного файла, все взаимодействие с сервером по сети, позволяет выделять для обработки одного блока данных несколько вычислительных узлов, объединенных в MPI-группу.

Реализация прикладного алгоритма сводится к написанию класса с заданным API, который будет обрабатывать данные, полученные от приложения и передавать приложению результат расчетов.

В случае использования режима, когда один блок данных обрабатывается несколькими MPI-процессами, один из них назначается ведущим и отвечает за взаимодействие с сервером, при этом, когда данные получены, то обработкой могут заниматься все процессы группы, включая ведущий.

Разработанная программная архитектура направлена на предоставление простого API для прикладных программистов, минимизацию нагрузки на процессор ЭУ и уменьшение требований к сетевому стеку операционной системы ЭУ путем выделения компоненты по диспетчеризации и передаче на большие расстояния потока данных в отдельный слой с возможностью его переноса на отдельный сервер.

Следующим этапом планируется проанализировать эффективность загрузки интерконнекта и сети ввода-вывода суперкомпьютера и проработать двухуровневый вариант передачи данных, когда в вычислительном поле выделяются некоторые узлы, которые занимаются только обменом данными с сервером и передают данные другим вычислительным узлам с использованием сети интерконнекта.

5. Оценка эффективности разработанного решения

Оценка эффективности проводилась путем измерения эффективной пропускной способности системы при передаче данных от экспериментальной установки через сервер очередей на площадке ИМСС УрО РАН (Пермь) на вычислительные узлы удаленного суперкомпьютера URAN ИММ УрО РАН (Екатеринбург). Коннективность связки между городами осуществлялась по выделенному vlan по DWDM магистрали «GIGA UrB RAS» с доступной пропускной способностью 1 Гбит/с и временем RTT=5,5 мс.

Исследовалось поведение системы при передаче измерений, состоящих из трех блоков с длинами 128, 1048576, 1048576 байт, соответственно. Измерения поступали от эмулятора ЭУ с промежутками между отправками данных, равными 20мс, 50мс и 100мс. Это приближенно соответствует нескольким режимам работы реальной ЭУ. При этом при минимальном интервале

скорость генерации составляет порядка 750Мбит/с, что меньше предельной пропускной способности канала связи и поэтому позволяет работать в реальном времени без бесконечного роста размера очереди исходных данных. Процесс на вычислительных узлах рассчитывал контрольные суммы блоков данных по алгоритму CRC32 и передавал их обратно на эмулятор ЭУ.

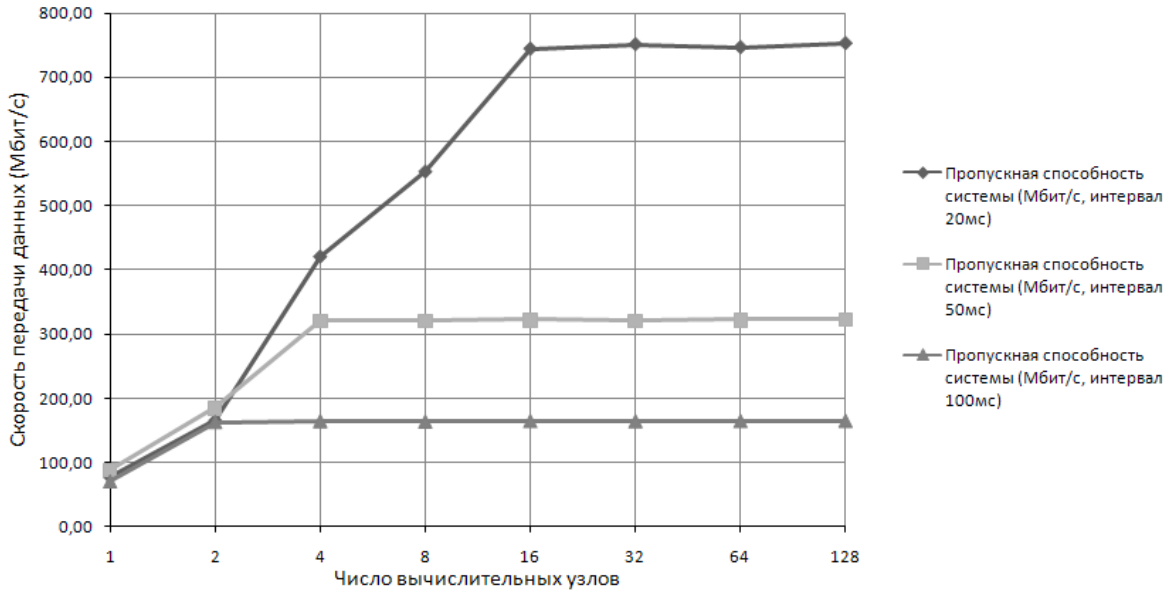


Рис. 8. Зависимости пропускной способности системы от числа задействованных вычислительных узлов

На рис. 8 показаны графики зависимости пропускной способности системы (Мбит/с) от числа задействованных вычислительных узлов. Для наглядности горизонтальная ось «число вычислительных узлов» приведена в логарифмическом масштабе.

Наклонные участки показывают рост пропускной способности системы с добавлением новых узлов, когда меньшее количество узлов не справляются с обработкой и небольшое число параллельных соединений не полностью утилизируют канал связи. При работе в этой области графика очередь растет, так как вычислительные узлы не успевают обрабатывать все данные в реальном времени.

Горизонтальные участки в правой части графика соответствуют установившемуся режиму, в котором скорость обработки данных совпадает со скоростью генерации данных. В этом случае увеличение числа задействованных вычислительных узлов не влияет на пропускную способность системы, но увеличивает ее надежность. Рост надежности обусловлен тем, что в случае отключения или потери сетевой связности части вычислительных узлов, а также в случае, если время обработки одного измерения по каким-либо причинам возрастет, то система все равно может остаться в рамках горизонтального участка графика и не допустить снижение пропускной способности ниже скорости генерации данных. Необходимо отметить, что разработанная архитектура позволяет, при необходимости, увеличивать число задействованных вычислительных узлов непосредственно в процессе работы системы, что предоставляет возможность оперативно реагировать на изменения условий проведения расчета.

6. Заключение

Предложена программная архитектура системы передачи интенсивного потока данных в распределенных системах. Сформулированы ограничения на класс прикладных задач, для которых возможно применение разработанного подхода. Спроектирован протокол, алгоритм диспетчеризации данных и разработано программное обеспечения для передачи данных с экспериментальной установки на узлы вычислительного кластера, тестирование которых подтверждает работоспособность предложенной программной архитектуры системы передачи интенсивного потока данных в распределенных системах.

Оценка эффективности показала увеличение пропускной способности системы при использовании разработанной технологии по сравнению с классическими подходами [2] и, как след-

ствие, уменьшение времени обработки массивов исходных данных. Показан возможный путь снижения нагрузки на процессор ЭУ путем переноса задачи диспетчеризации на близкорасположенный выделенный сервер. Помимо этого, использование выделенного сервера позволяет проводить сборку набора данных из нескольких несвязанных между собой источников, например, при наличии нескольких независимых групп датчиков, допускающих раздельную обработку данных с них.

Разработанная технология предоставляет принципиально новый инструмент проведения экспериментальных исследований, позволяя обрабатывать быстропротекающие процессы в течение длительного времени, например, при лабораторном изучении начальной стадии формирования тропических циклонов.

Дальнейшие исследования будут направлены на детальную разработку методики оценки необходимого вычислительного ресурса, исходя из параметров эксперимента, расчетного алгоритма и пропускной способности сети, а также на совершенствование программного обеспечения для оптимального использования ресурсов внутренних сетей суперкомпьютера, таких, как сеть ввода вывода и сеть передачи сообщений MPI.

Автор выражает благодарность научному руководителю исследования Григорию Федоровичу Масичу (зав. лабораторией телекоммуникационных и информационных систем ИМСС УрО РАН) за помощь в проведении исследований.

Литература

1. Степанов Р.А., Масич А.Г., Сухановский А.Н., Щапов В.А., Игумнов А.С., Масич Г.Ф. Обработка на супервычислителе потока экспериментальных данных // Вестник УГАТУ, Уфа, 2012. Т. 16, № 3 (48). С. 126-133.
2. Р.А. Степанов, А.Г. Масич, Г.Ф. Масич «Инициативный проект «Распределенный PIV»» // Научный сервис в сети Интернет: масштабируемость, параллельность, эффективность: труды Всероссийской суперкомпьютерной конференции – М.: Изд-во МГУ, 2009. – С. 360–363. (ISBN 978-5-211-05697-8).
3. А.Г. Масич, Г.Ф. Масич «Инициатива GIGA UrB RAS» // Совместный вып. Журнала «Вычислительные технологии» и журн. «Вестник КазНУ им. Аль-Фараби». Сер. «Математика, механика, информатика», №3 (58). По материалам Междунар. конф. «Вычислительные и информационные технологии в науке, технике и образовании», - Казахстан, Алматы.-2008.- Т.13.- Ч. II. -С. 413-418 (ISSN 1560-7534).
4. А.Г. Масич, Г.Ф. Масич. GIGA UrB RAS подход к LambdaGrid парадигмам вычислений // Научный сервис в сети Интернет: суперкомпьютерные центры и задачи: Труды Международной суперкомпьютерной конференции (20-25 сентября 2010 г., г. Новороссийск). – М.: Изд-во МГУ, 2010. С. 4-11.
5. А.Г. Масич, Г.Ф. Масич, Р.А. Степанов, В.А. Щапов Скоростной I/O-канал супервычислителя и протокол обмена интенсивным потоком экспериментальных данных // Материалы X международной конференции «Высокопроизводительные параллельные вычисления на кластерных системах HPC-2010» – Пермь: Изд-во ПГТУ, 2010. - Т. 2. С. 119–128. (ISBN 978-5-398-00506-6).
6. Щапов В.А., Масич А.Г., Масич Г.Ф. Модель потоковой обработки экспериментальных данных в распределенных системах // Вычислительные методы и программирование. 2012. Раздел 2. 139-145 (http://num-meth.srcc.msu.ru/).
7. Yunhong Gu and Robert L. Grossman, UDT: UDP-based Data Transfer for High-Speed Wide Area Networks, Computer Networks (Elsevier). Volume 51, Issue 7. May 2007.
8. Vladislav Shchapov, Alexey Masich. Protocol of High Speed Data Transfer from Particle Image Velocimetry System to Supercomputer // Proc. of The 7th International Forum on Strategic Technology (IFOST 2012) September 18-21, 2012, Vol.1. National Research Tomsk Polytechnic University, Tomsk, P. 653-657