УДК 004.457, 004.382.2 DOI: 10.14529/cmse150103

ОБ ОДНОМ ПОДХОДЕ К МОДЕЛИРОВАНИЮ СУПЕРКОМПЬЮТЕРНЫХ КОМПЛЕКСОВ[1]

П.А. Швец, Вад.В. Воеводин, С.И. Соболев

В НИВЦ МГУ предложен подход к созданию системы контроля автономного функционирования суперкомпьютерных комплексов на основе графовой модели суперкомпьютера. С использованием данного подхода была реализована система контроля Octotron, которая сейчас проходит апробацию в суперкомпьютерном центре МГУ. Данная статья описывает проблемы и задачи, с которыми столкнулись авторы при реализации данной системы и ее запуске на суперкомпьютерах «Чебышёв» и «Ломоносов». Рассматриваются выбранные и разработанные авторами программные инструменты для работы с графами, кратко описывается язык, используемый для описания модели, затрагиваются вопросы визуализация модели и импорта данных мониторинга.

Ключевые слова: суперкомпъютер, модель суперкомпьютера, мониторинг, инструменты программирования, автономное функционирование, надежность.

Введение

Поддержка автономного функционирования суперкомпьютерного центра — одна из важнейших задач, с которой сталкиваются их держатели и администраторы. Этот же вопрос встал перед нами в рамках работ по обеспечению эффективной надежной работы Суперкомпьютерного комплекса МГУ. Однако анализ мировой практики поддержки суперкомпьютерных центров показал, что каждый решает эту задачу по-своему, создавая индивидуальные и непереносимые комплексы.

Нами был предложен [1] метод контроля работы суперкомпьютерного комплекса на основе модели его функционирования, представленной в виде расширенного мультиграфа. Вершины графа описывают физические (ЦПУ, кондиционеры др.) и логические (область подкачки, файловая система и др.) компоненты комплекса. Ребра графа описывают связи между компонентами: например, вершина «стойка» может быть связана с вершиной «шасси» связью «содержит», вершина «кондиционер» — связана с вершиной «горячий коридор» связью «охлаждает». Пример простейшей модели показан на рис. 1. С вершинами и связями модели ассоциируются атрибуты, описывающие состояние компонент (температура, IP-адрес и др.), правила, позволяющие преобразовывать атрибуты (например, вычислять скорость изменения характеристик) и реакции, срабатывающие, когда атрибуты принимают определенное значение (например, информирование системного администратора). Разработанная нами система контроля, основываясь на описанной модели, производит получение, обработку и анализ реальных данных с различных аппаратных и программных сенсоров, а с помощью набора правил и реакций происходит контролирование штатной работы комплекса. Данный подход позволяет обрабатывать ряд ситуаций, которые сложно отслеживать в рамках раздельного мониторинга и контроля всех компонент, при котором не учитываются физические и логические связи между разными компонентами. Например, при проблемах с одним источником питания можно от-

[1] Статья рекомендована к публикации программным комитетом Международной суперкомпьютерной конференции «Научный сервис в сети Интернет: многообразие суперкомпьютерных миров».

ключить только то оборудование, которое питается непосредственно от него, или же следить за количеством узлов в конкретной очереди с ошибками определенного типа. Также наличие модели со связями полезно для систематизации знаний о суперкомпьютерных комплексах и потенциальных источниках проблем. Каждый объект рассматривается как потенциально сбойный. Становится возможным оценить, как сбой каждого конкретного компонента повлияет на соседние компоненты и на работу комплекса в целом. Подробней принципы моделирования описаны в статье [1]. Настоящая работа посвящена детальному описанию реализованного нами подхода к моделированию суперкомпьютерных комплексов.

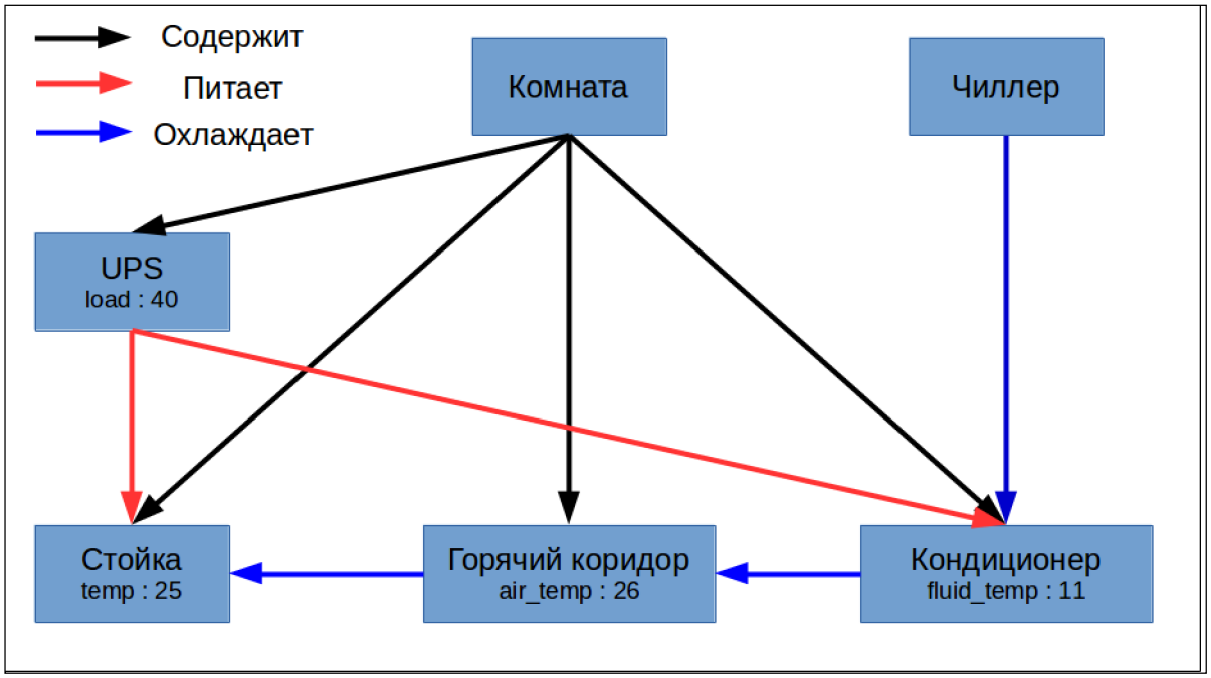

Рис. 1. Пример части модели суперкомпьютера

Статья организована следующим образом. В разделах 1 и 2 рассматриваются высокоуровневые и низкоуровневые инструменты для работы с графами. В разделах 3 и 4 описываются и обосновывается выбранный нами подход к разработке комплекса. В разделе 5 описывается используемый язык описания модели и приводятся примеры описания. В 6 разделе рассматривается методика построения полной модели суперкомпьютера. Раздел 7 посвящен вопросу визуализации построенной модели суперкомпьютера. В разделе 8 описан подход к управлению системой. Раздел 9 посвящен настройке средств мониторинга на суперкомпьютерах «Чебышёв» и «Ломоносов» для использования в разработанной системе контроля. В заключении суммируются основные результаты работы и описываются планы дальнейшего развития системы.

1. Анализ средств для работы с графами

На основе изучения характеристик современных суперкомпьютерных комплексов и формирования базовых принципов построения их моделей были сформулированы требования к программной системе работы с графами, которая должна лежать в основе разрабатываемого нами программного средства:

- Быстрая работа с большими графами. Сотни тысяч вершин, миллионы атрибутов — таков, по нашей оценке, масштаб требуемых графов для описания моделей суперкомпьютеров верхних строчек списка Top500.
- Поддержка кратных ребер. Некоторые объекты модели могут быть связаны двумя и более типами связей. Например, вершина «стойка» должна быть связана с вершиной «шасси» связями «содержит» и «питает».
- Поддержка атрибутов разных типов (строки, целые числа, числа с плавающей запятой) для вершин и ребер.

Нами были изучены известные средства для работы с большими графами, такие как Gephi [2] и Tulip/porgy [3].Gephi больше ориентирован на интерактивную работу пользователя с графами, а Tulip/porgy, имеющий средства для автоматического преобразования графов, использовал неподходящую нам систему правил преобразований. Доработка этих инструментов была признана нецелесообразной. Было принято решение об исследовании более низкоуровневых средств.

2. Анализ низкоуровневых библиотек и программных средств для работы с графами

Существует огромное множество библиотек для работы с графами на самых разных языках программирования, поэтому мы рассмотрели наиболее популярные из них и выбрали удовлетворяющую нашим требованиям.

- Boost Graph Library [4] (BGL) — популярная библиотека, отвергнутая нами из-за ее основного языка программирования — C++. Требования к разрабатываемой системе корректировались в ходе разработки, что могло бы потребовать очень больших временных затрат при использовании C++ как основного языка проекта.
- Graph-tool [5] — интерфейс для BGL на языке программирования Python. Согласно предварительной оценке, производительности языка не хватило бы для обработки требующихся нам объемов данных.
- Neo4j [6] — графовая СУБД, написанная на языке Java и ориентированная на работу с высоконагруженными сервисами.

3. Технологии реализации

Для разработки нами был выбран язык Java, к достоинствам которого можно отнести возможность быстрой разработки и наличие множества готовых библиотек, и база данных Neo4j как средство для работы с графами. Neo4j — нереляционная база данных, оперирующая понятиями «узлы» (nodes), «связи» (relationships) и «атрибуты» (properties), которые могут устанавливаться как на узлы, так и на связи. Это полностью подходит под наши требования к описанию модели. Кроме того, Neo4j обеспечивает следующие полезные возможности:

- Механизм автоматического кэширования позволяет хранить часто используемые данные в памяти, а редко используемые — на диске. Данная особенность позволяет эффективно работать с графами очень больших размеров, что может понадобиться для описания суперкомпьютеров экзафлопсного масштаба.
- Быстрый поиск по свойствам (индексирование) напрямую влияет на скорость исполнения запросов к системе, а значит, и на максимальное количество обрабатываемых запросов в секунду, позволяя обрабатывать данные от большего числа сенсоров с большей частотой.

- ACID-транзакции [7] — данная модель транзакций облегчает техническую обработку параллельных запросов.
- Готовый механизм сохранения и загрузки с диска позволяет продолжать работу системы после остановки без потери прошлых данных.

4. Интерфейс работы с графами (API)

Мы предусмотрели возможность для работы не только с Neo4j, реализовав свой промежуточный интерфейс для работы с графами и используя Neo4j как одну из возможных реализаций. Если в какой-то момент нам понадобится функционал, отсутствующий в Neo4j, но присутствующий в другой системе, мы сможем перейти на нее без существенного изменения логики работы системы. Исходя из наших требований, API для работы с графом реализует следующие возможности:

- Создать объект.
- Создать связи между созданными объектами.
- Установить и получить атрибуты по объекту или связи.
- Удалить заданные объекты, связи или атрибуты.

API для поиска по графу реализует следующие возможности:

- Найти все объекты/связи, у которых есть атрибут с заданным именем.
- Найти все объекты/связи, у которых есть атрибут с заданным именем и значением.
- Найти все объекты/связи, у которых есть строковые атрибуты с заданным именем и значением, подходящим под заданный шаблон.

На основе этих двух API мы реализовали все внутренние сервисы, позволяющие сохранять в модели более сложные объекты, обеспечивающие возможность обновления атрибутов, правил, вызов реакций, и реализующие внешний протокол запросов к графу.

5. Язык описания модели

После разработки системы встал вопрос об описании самой модели. Наиболее простой метод — описание модели на языке Java с помощью разработанного API. К сожалению, этот подход имеет существенный недостаток: Java — довольно «многословный» язык, а для описания модели хватило бы лишь небольшого подмножества языка. Также сама необходимость знать и использовать для описания язык Java может восприниматься неоднозначно.

Была предпринята попытка разработки собственного языка описания модели, который бы потом транслировался в Java. Первая версия преобразовывала код построчно с помощью набора регулярных выражений, но вскоре стало понятно, что ни развивать, ни поддерживать такой вариант невозможно: требовались все более сложные языковые конструкции, а их реализация с помощью регулярных выражения получалась очень сложной, громоздкой и неэффективной.

На следующем этапе мы изучили готовые средства, генерирующие трансляторы по формальной грамматике языка (например, ANTLR [8]), но после исследования наработок по этому направлению решили, что полноценная реализация собственного описательного языка займет слишком много времени.

В итоге в качестве основного языка описания моделей был выбран язык Python [9] и использован специальный интерпретатор Jython [10], исполняющий код на JVM (виртуальная машина, на которой исполняются Java-программы) и позволяющий без каких-либо дополнительных усилий использовать Java-классы из программы, написанной на Python.

Мы разработали модуль, предоставляющий простой интерфейс для исходного API на языке Java, и предоставляем пользователям примеры и документацию для создания своих моделей.

Рассмотрим, как с помощью предложенного подхода создаются модели. Первая часть описания модели — задание атрибутов, правил и реакций.

- Описание атрибутов совпадает с заданием словаря на языке Python в формате «имя : значение» для константных атрибутов или «имя : тип (значение по умолчанию)» для изменяющихся атрибутов (сенсоров). Значение по умолчанию может отсутствовать и тогда атрибут будет использоваться только после первого обновления. Атрибуты могут быть целыми и вещественными числами, строками и булевыми константами True/False.
- Правила описываются в стиле словаря, в формате «имя : конструктор правила». Имя обозначает имя атрибута, который будет создаваться правилом. Конструкторы правил индивидуальны в каждом случае и полностью описаны в документации [14].
- Реакции описываются в стиле словаря, в формате «условие реакции : конструктор реакции». Реакция срабатывает, когда выполняется требуемое условие, наиболее часто используемое условие - когда соответствующий атрибут принимает указанное значение. В качестве реакции мы создали 4 стандартных типа (Info, Warning, Danger, Critical), но у пользователя есть возможность модифицировать реакции, добавляя вызов произвольных скриптов или создавая собственные реакции.

```
eth_module : {
  "const" : {
    "interface" : "eth0"
  }
  "sensor" : {
    "recieve_errors" : Long(),
    "duplex" : String()
  }
  "var" : {
    "error_speed" : Speed("recieve_errors"),
    "low_errors" : UpperThreshold("error_speed", 10)
  }
  "react" : {
    Eqauls("low_errors", False).Delay(60) :
      Warning("tag", "NETWORK")
        .Msg("descr", "{interface}: errors growing fast in last minute"),
    NotEquals("duplex", "full") :
      Danger("tag", "NETWORK") :
        .Msg("descr", "{interface}: duplex mode changed: {duplex}")
  }
}
```

Рис. 2. Пример описания модуля для проверки сетевого интерфейса

Набор атрибутов, правил и реакций может быть объединен в модуль. Модули переносимы и могут быть использованы для создания нескольких разных моделей, имеющих схожие части. Мы предоставляем пользователям набор готовых модулей, соответствующих некоторым популярным средствам и протоколам мониторинга (SNMP, collectd). Пример описания модуля приведен на рис. 2. Данный модуль проверяет, что режим интерфейса (duplex) соответствует требуемому и что ошибки приема не растут быстрее, чем 10

штук в секунду на протяжении минуты (небольшой рост или кратковременные всплески могут присутствовать, но они не является опасным).

Вторая часть описания модели — создание объектов и связей.

- Для создания объектов используется функция CreateObjects(count, modules...) и CreateObject(modules...). Параметрами данных функций являются модули из описанных выше атрибутов, правил и реакций, а также количество объектов, которые необходимо создать. Есть возможность описывать необходимые модули в самом вызове функции. Первая функция возвращает список объектов, а вторая — один объект.
- Для создания связей между объектами используется набор функций OneToOne, OneToEvery, AllToAll и другие (описание каждой функции доступно в документации [14]). Параметрами этих функций являются сущности, которые надо соединить, и список типов связей.

```
# создание объектов
room      = CreateObject()
chiller   = CreateObject()
ups       = CreateObject({"sensor" : {"load" : Long()}})
fan       = CreateObject({"sensor" : {"fluid_temp" : Long()}})
hot_aisle = CreateObject({"sensor" : {"air_temp" : Long()}})
rack      = CreateObject({"sensor" : {"temp" : Long()}})

# создание связей между компонентами
OneToOne(room, ups      , "contains")
OneToOne(room, fan      , "contains")
OneToOne(room, hot_aisle, "contains")
OneToOne(room, rack     , "contains")
OneToOne(room, chiller  , "contains")
OneToOne(room, chiller  , "contains")

OneToOne(ups, fan , "power")
OneToOne(ups, rack, "power")

OneToOne(chiller  , fan      , "chill")
OneToOne(fan      , hot_aisle, "chill")
OneToOne(hot_aisle, rack     , "chill")
```

Рис. 3. Описание модели, изображенной на рис. 1, с помощью разработанного API

На рис. 3 приведен пример описания модели, соответствующей фрагменту, изображенному на рис. 1.

6. Методика построения модели

Суперкомпьютеры состоят из множества различных компонент, что существенно затрудняет построение полной описывающей модели. Однако чем больше компонент будет внесено в такую модель, тем более полный контроль над комплексом будет у системы.

На данный момент мы описываем следующие подсистемы суперкомпьютера (в скобках указаны примеры объектов из этой подсистемы):

- Система электропитания (источники бесперебойного питания, модули с батареями).
- Система охлаждения (чиллеры, кондиционеры, мониторинг среды).
- Управляющая часть (узлы доступа и компиляции, очереди задач).
- Вычислительная часть (шасси, узлы, диски, память).

- Файловая система (зависит от типа ФС).
- Ethernet сеть (коммутаторы, порты).
- Infiniband сеть (коммутаторы, менеджер сети).

Модель можно описывать вручную или же частично генерировать программно, пользуясь довольно регулярной структурой некоторых компонент. Индивидуальные данные объектов (ip-адрес, серийный номер и др.) в таком случае подгружаются из внешних CSV-файлов.

Одним из направлений работы над проектом является разработка инструментария для автоматизированного построения модели [15]. Сгенерированная модель будет требовать дальнейшей ручной доработки, но часть рутинной работы будет уже выполнена.

7. Визуализация модели

В ходе разработки регулярно возникала необходимость визуально оценить созданную модель. Изображение модели может быть полезно как для эмпирической проверки корректности построения, так и для демонстрационных или обучающих целей. Однако полностью отобразить модель практически нереально — алгоритмы расположения графов на плоскости не справляются с графами такого масштаба или показывают совершенно нечитаемый результат.

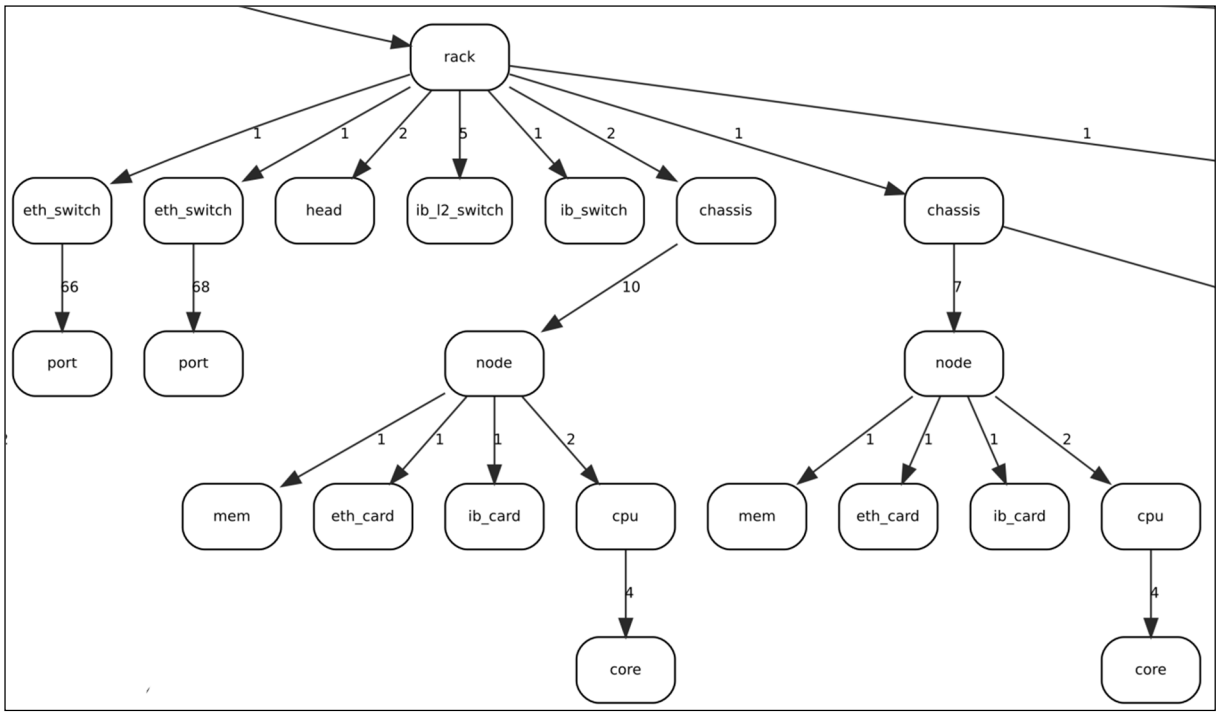

Рис. 4. Пример визуализации агрегированной модели

Но если не ставить задачу визуализации полного графа, а использовать регулярность многих компонент графа, то задача визуализации имеет решение. По многим типам связей модель выглядит как дерево с наличием похожих поддеревьев, которые можно разбивать по уровням и группировать, отображая в итоге метаграф из сгруппированных подграфов.

Был разработан прототип веб-сервиса, который производит «агрегацию» объектов модели и отображает получившийся метаграф. Пример результата его работы можно видеть на рис. 4 — это визуализация части модели суперкомпьютера «Чебышёв» [11].

Цифры на связях показывают, сколько «похожих» поддеревьев было объединено в одну вершину.

8. Работа с системой

После описания модели можно произвести запуск самой системы контроля и настроить импорт реальных данных. Управление системой происходит с помощью http-запросов к разработанной компоненте. Данный метод используется и для автоматического импорта данных в модель и для ручного управления. Такой подход позволяет управлять всем мониторингом из обычного веб-браузера и может легко интегрироваться в другие веб-сервисы, например, для визуализации текущего состояния суперкомпьютерного комплекса.

9. Наполнение системы реальными данными

Для нужд оперативного контроля информацию от суперкомпьютера можно распределить по трем категориям:

1. Информация от инфраструктурного оборудования — охлаждение, питание и прочее. Такой информации достаточно мало, однако именно она требует наиболее оперативной обработки и реагирования для сохранения оборудования комплекса в целостности. Требуемое время оповещения и реагирования — минуты. Также здесь используется механизм «активных событий» — при некоторых событиях данные в модель заносятся сразу при наступлении события, в обход стандартного механизма опроса.
2. Служебная информация от основных систем суперкомпьютера — файловая система, система очередей, служебные узлы. Данной информации значительно больше, ошибки могут повлечь проблемы с доступом к суперкомпьютеру, но не могут навредить оборудованию. Требуемое время оповещения и реагирования — от нескольких минут, до десятков минут.
3. Служебная информация с узлов — локальные проблемы на узлах, выход узлов из строя. Ошибки, которые могут проявляться на этом уровне, повлияют лишь на некоторые пользовательские задачи и не могут нанести серьезный вред оборудованию. Требуемое время оповещения и реагирования — десятки минут, часы.

Поток информации первой и второй категории не превысит нескольких сотен обновлений атрибутов в секунду, а поток информации третей категории не имеет фиксированных требований к частоте съема, которую можно регулировать для стабильной работы системы.

Таблица

Частота съема данных мониторинга

	«Чебышёв» (≈600 узлов)	«Ломоносов» (≈5000 узлов)
1-я категория	Опрос 1 раз в минуту Активные события	Опрос 1 раз в минуту Активные события
2-я категория	Опрос 1 раз в минуту	Опрос 1 раз в минуту
3-я категория	Опрос 1 раз в минуту	Опрос 1 раз в 10 минут

На данный момент система, работающая на среднем компьютере и графе, помещающемся в память, обрабатывает около 15 тысяч обновлений атрибутов в секунду. Исходя из этих данных, на системах «Чебышёв» и «Ломоносов» суперкомпьютерного комплекса МГУ мы установили частоту съема данных, указанную в таблице.

Заключение

В статье рассмотрен и обоснован выбор программных инструментов для работы с графами. Описан предложенный подход к описанию моделей суперкомпьютеров, предложен новый метод визуализации граф, получаемых из моделей. В статье описаны детали настройки разработанной системы на реальных суперкомпьютерах.

Система внедрена в опытную эксплуатацию в Суперкомпьютерном комплексе МГУ. Разработанная система доступна под открытой MIT лицензией [13, 14].

В дальнейшем мы планируем развивать средства визуализации и автоматической генерации модели и расширить библиотеку стандартных компонент суперкомпьютеров и средств мониторинга для облегчения процесса создания модели.

Работа выполнена при финансовой поддержке РФФИ, грант №12-07-33047.

Литература

1. Антонов, А.С. Разработка принципов построения и реализация прототипа системы обеспечения оперативного контроля и эффективной автономной работы суперкомпьютерных комплексов / А.С. Антонов, В.В. Воеводин, Вад.В. Воеводин и др. // Вестник УГАТУ. — 2014. — Т. 18, № 2. — С. 227–236.
2. Bastian, M. Gephi: an open source software for exploring and manipulating networks. / M. Bastian, S. Heymann, M. Jacomy // International AAAI Conference on Weblogs and Social Media. — 2009. — Vol. 8. — P. 361–362.
3. Pinaud, B. PORGY: A Visual Graph Rewriting Environment for Complex Systems. / B. Pinaud, G. Melançon, J. Dubois // Computer Graphics Forum — Eurographics Conference on Visualization (EuroVis 2012) special issue. — 2012. — Vol. 31. — P. 1265–1274.
4. BGL Library Documentation. URL: http://boost.org/libs/graph/doc/ (дата обращения: 29.12.2014).
5. GraphTool Program Description. URL: http://graph-tool.skewed.de/ (дата обращения: 29.12.2014).
6. Neo4j DBMS Description. URL: http://neo4j.org (дата обращения: 29.12.2014).
7. Gray, J. The Transaction Concept: Virtues and Limitations. / J. Gray. // Proceedings of the 7th International Conference on Very Large Databases. — 1981. — P. 144–154.
8. ANTLR Parser Generator Description ANTLR. URL: http://antlr.org (дата обращения: 29.12.2014).
9. Python Programming Language Description. URL: http://python.org (дата обращения: 29.12.2014).
10. Jython Interpreter Description. URL: http://jython.org (дата обращения: 29.12.2014).
11. Антонов, А.С. СКИФ МГУ — основа Суперкомпьютерного комплекса Московского университета / А.С. Антонов // Вторая Международная научная конференция «Суперкомпьютерные системы и их применение»: доклады конференции (27–29 октября 2008, Минск). — ОИПИ НАН Беларуси, 2008. — С. 7–10.

12. Воеводин, В.В. Практика суперкомпьютера «Ломоносов» / В.В. Воеводин, С.А. Жуматий, С.И. Соболев и др. // Открытые системы. — 2012. — № 7. — С. 36–39.
13. Ядро системы Octotron. URL: https://github.com/srcc-msu/octotron_core (дата обращения: 29.12.2014).
14. Рабочее окружения для создания модели системы Octotron. URL: https://github.com/srcc-msu/octotron (дата обращения: 29.12.2014).
15. Воеводин, Вад.В. Автоматическое определение и описание сетевой инфраструктуры суперкомпьютеров / В.В. Воеводин, К.С. Стефанов // Вычислительные методы и программирование: Новые вычислительные технологии. — 2014. — Т. 15, № 3. — С. 560–568.

Швец Павел Артёмович, программист, Научно-исследовательский вычислительный центр, Московский государственный университет имени М.В. Ломоносова (Москва, Российская Федерация), shvets.pavel.srcc@gmail.com

Воеводин Вадим Владимирович, к.ф.-м.н., научный сотрудник, Научно-исследовательский вычислительный центр, Московский государственный университет имени М.В. Ломоносова (Москва, Российская Федерация), vadim@parallel.ru

Соболев Сергей Игоревич, к.ф.-м.н, старший научный сотрудник, Научно-исследовательский вычислительный центр, Московский государственный университет имени М.В. Ломоносова (Москва, Российская Федерация), sergeys@parallel.ru

Поступила в редакцию 26 декабря 2014 г.

Bulletin of the South Ural State University
Series "Computational Mathematics and Software Engineering"
2015, vol. 4, no. 1, pp. 33–43

DOI: 10.14529/cmse150103

AN APPROACH TO MODELING OF SUPERCOMPUTING CENTERS

P.A. Shvets, Research Computing Center, Moscow State University (Moscow, Russian Federation) shvets.pavel.srcc@gmail.com,
Vad.V. Voevodin, Research Computing Center, Moscow State University (Moscow, Russian Federation) vadim@parallel.ru,
S.I. Sobolev, Research Computing Center, Moscow State University (Moscow, Russian Federation) sergeys@parallel.ru

An approach to implementation of supercomputing center control system based on supercomputer graph model has been proposed in RCC MSU. The Octotron system has been developed on the basis of this approach, which is being tested in MSU Supercomputing Center currently. The article describes challenges and tasks, encountered by authors while developing and running the system on supercomputers «Chebyshev» and «Lomonosov». It also includes overview of graph tools used, brief description of modeling language, model visualization and monitoring data import.

Keywords: supercomputer, supercomputer model, monitoring, programming tools, autonomous functioning, resiliency.

References

1. Antonov A.S., Voevodin V.V., Voevodin Vad.V. Razrabotka principov postroenija i realizacija prototipa sistemy obespechenija operativnogo kontrolja i jeffektivnoj avtonomnoj raboty su-perkomp'juternyh kompleksov [Principles of Development System for Operational Control and Efficient Autonomous Work of Supercomputers]. Vestnik UGATU [Bulletin of UGATU]. 2014. Vol. 18. No. 2. P. 227–236.
2. Bastian M., Heymann S., Jacomy M. Gephi: an open source software for exploring and manipulating networks. International AAAI Conference on Weblogs and Social Media. 2009. Vol. 8. P. 361–362.
3. Pinaud B., Melançon G., Dubois J. PORGY: A Visual Graph Rewriting Environment for Complex Systems. Computer Graphics Forum — Eurographics Conference on Visualization (EuroVis 2012) special issue. 2012. Vol. 31. P. 1265–1274.
4. BGL Library Documentation. URL: http://boost.org/libs/graph/doc/ (accessed: 29.12.2014).
5. GraphTool Program Description. URL: http://graph-tool.skewed.de/ (accessed: 29.12.2014).
6. Neo4j DBMS Description. URL: http://neo4j.org (accessed: 29.12.2014).
7. Gray J. The Transaction Concept: Virtues and Limitations. Proceedings of the 7th International Conference on Very Large Databases. 1981. P. 144–154.
8. ANTLR Parser Generator Description. URL: http://antlr.org (accessed: 29.12.2014).
9. Python Programming Language Description. URL: http://python.org (accessed: 29.12.2014).
10. Jython Interpreter Description. URL: http://jython.org (accessed: 29.12.2014).
11. Antonov A.S. SKIF MGU - osnova Superkomp'juternogo kompleksa Moskovskogo universiteta [SKIF MSU — the Foundation of Moscow University Supercomputing Center]. Vtoraja Mezhdunarodnaja nauchnaja konferencija «Superkomp'juternye sistemy i ih primenenie»: doklady konferencii (27-29 oktjabrja 2008, Minsk) [Second Internation Conference «Supercomputing Systems and Applications»: papers (27–29 October 2008, Minsk)]. OIPI NAN Belarusi, 2008. P. 7–10.
12. Voevodin V.V., Zhumatij S.A., Sobolev S.I. Praktika superkomp'jutera «Lomonosov» [Practice of «Lomonosov» Supercomputer]. Otkrytye sistemy [Open Systems]. 2012. No. 7. P. 36–39.
13. Jadro sistemy Octotron [Octotron System Core]. URL: https://github.com/srcc-msu/octotron_core (accessed: 29.12.2014).
14. Rabochee okruzhenija dlja sozdanija modeli sistemy Octotron [Octotron Model Creation Environment]. URL: https://github.com/srcc-msu/octotron (accessed: 29.12.2014).
15. Vad.V. Voevodin, K.S. Stefanov. Avtomaticheskoe opredelenie i opisanie setevoj infrastruktury superkomp'juterov [Automatic Detection and Description of a Supercomputer Network Infrastructure]. Vychislitel'nye metody i programmirovanie: Novye vychislitel'nye tehnologii [Computational methods and programming: new computational technologies]. 2014. Vol. 15, No. 3. P. 560–568.

Received December 26, 2014.