РАЗРАБОТКА ПРИНЦИПОВ ПОСТРОЕНИЯ И РЕАЛИЗАЦИЯ ПРОТОТИПА СИСТЕМЫ ОБЕСПЕЧЕНИЯ ОПЕРАТИВНОГО КОНТРОЛЯ И ЭФФЕКТИВНОЙ АВТОНОМНОЙ РАБОТЫ СУПЕРКОМПЬЮТЕРНЫХ КОМПЛЕКСОВ

А.С. Антонов, Вад.В. Воеводин, Вл.В. Воеводин, С.А. Жуматий, Д.А. Никитенко, С.И. Соболев, К.С. Стефанов, П.А. Швец

Введение

Устойчивый рост производительности суперкомпьютерных комплексов, как уже существующих, так и только проектируемых, сопровождается неизбежным увеличением числа и сложности их компонентов. Это относится как к оборудованию, включающему в себя вычислительные и коммуникационные модули, подсистемы охлаждения, энергообеспечения и т.д., так и к программной инфраструктуре, на которую ложится существенная задача по поддержке эффективной согласованной работы всех компонентов комплекса. Чем сложнее система, чем больше компонентов в ней, тем сложнее решается задача обеспечения ее надежности и корректного и эффективного функционирования. Эта проблема в настоящее время актуальна для всех суперкомпьютерных комплексов, обслуживающих сотни и тысячи пользователей, которые должны быть обеспечены доступом к высокопроизводительным ресурсам в режиме 24/7.

Актуальность проблемы определяется как существенной стоимостью суперкомпьютерных систем, так и их высокой востребованностью вычислительным сообществом. Стоимость суперкомпьютеров значительна, спрос на суперкомпьютерные ресурсы велик, значит, необходимо увеличивать их отдачу, минимизируя простои оборудования, отдавая в каждый момент в работу все то, что можно отдать пользователям для расчетов. Если какая-то часть суперкомпьютера неисправна, ее нужно локализовать и вывести из конфигурации, оставив в работе максимально возможную часть нормально функционирующего оборудования для продолжения обслуживания потока задач.

Вместе с этим, современные суперкомпьютерные комплексы обладают двумя критически важными свойствами, определяющими сложность их построения и последующего сопровождения: чрезвычайно большое число компонентов и высокий уровень энергопотребления.

Первое свойство – большое число компонентов – приводит к тому, что в суперкомпьютерах постоянно какие-то компоненты выходят из строя. Неисправные компоненты будут всегда, поэтому их негативное влияние на работу всего суперкомпьютерного комплекса должно быть оперативно минимизировано, а машина должна продолжать эффективно работать. Неисправные узлы нужно уметь оперативно обнаруживать и выводить из счетного поля для обеспечения эффективной работы суперкомпьютерного комплекса в целом. Важно, что подобного рода работа должна выполняться постоянно и автоматически.

Второе свойство современных суперкомпьютерных комплексов – высокий уровень энергопотребления в совокупности с высокой плотностью размещения компонент – приводит к необходимости построения серьезных систем охлаждения и постоянного оперативного отвода большого объема тепла. Сбой в системах охлаждения может привести не только к снижению эффективности работы суперкомпьютерного комплекса или к выходу из строя части дорогостоящего оборудования, но и к возникновению опасных аварийных ситуаций. Например, технология «горячих коридоров», используемая во многих суперкомпьютерных центрах, требует постоянного внимания к своей работе: в случае отключения систем охлаждения температура внутри горячего коридора будет повышаться примерно на 20 градусов каждую минуту, что уже через несколько минут безвозвратно погубит все оборудование. В такой ситуации отсутствие контроля за функционированием суперкомпьютерного комплекса даже в течение 5 минут может иметь самые серьезные и необратимые последствия. Нужна постоянно работающая система, которая должна отслеживать большое число аппаратных датчиков (температуры, влажности, утечки охлаждающей жидкости и т.п.), самостоятельно принимающая решение относительно необходимости отключения отдельных компонент, подсистем, а иногда и всего суперкомпьютерного комплекса.

Очень важный момент – полнота охвата такой системой всего спектра возможных аварийных ситуаций. В ней должен найти отражение весь предшествующий опыт сопровождения суперкомпьютерных комплексов. Это позволит не совершать два раза одних и тех же ошибок, накапливая как собственный опыт, так и опыт других коллективов в сопровождении суперкомпьютерных систем. А опыт уже богат, и примеров тому много. Датчик протечки охлаждающей жидкости может молчать и не сигнализировать о проблеме не потому, что все в порядке, а потому, что он вышел из строя. Система оповещения по SMS может не присылать тревожных сигналов не потому, что все в порядке, а потому, что на счету просто закончились деньги. Аварийное отключение систем охлаждения было вовремя замечено автоматикой, но оказалось, что после последней перезагрузки головного узла суперкомпьютера служебный процесс, отвечающий за выключение оборудование, не стартовал… Подобных ситуаций очень много, все они потенциально опасны и требуют незамедлительной реакции.

Традиционные методы обеспечения надежности суперкомпьютерного комплекса заключаются в выборе, настройке и использовании одной из доступных систем мониторинга и визуализации состояния комплекса (Nagios [1], Ganglia [2], Zabbix [3], Zenoss [4] и др.). Мониторинг инфраструктурного оборудования может обеспечиваться программными решениями его производителей. Наконец, для учета специфических особенностей комплекса и определения наиболее опасных или характерных аварийных ситуаций создается набор сценариев («скриптов»), которые срабатывают либо периодически, либо по наступлению какого-либо события, и выполняют ряд специализированных проверок (доступна ли очередь заданий, все ли вычислительные узлы находятся в готовности и т.д.).

Однако такой подход ориентирован прежде всего на сбор информации о состоянии множества отдельных компонентов вычислительного комплекса, в основном относящихся к его вычислительным модулям, и их минимальную агрегацию и визуализацию. Логика по выявлению сбоев и реагированию на аварийные ситуации фактически отдается на откуп системным администраторам комплекса, которые и реализуют ее в виде набора дополнительных программных модулей. Подобные решения практически не являются переносимыми между различными вычислительными комплексами, они не поддерживают накопление истории возникновения критических ситуаций и ее анализ, формирование банков аварий и методов реагирования на них.

О проекте

В начале 2013 г. в НИВЦ МГУ имени М.В. Ломоносова стартовал проект по созданию принципов построения и реализации системы обеспечения оперативного контроля и эффективной автономной работы суперкомпьютерных комплексов. Разработанную систему планируется внедрить и использовать на имеющихся в МГУ суперкомпьютерах «Чебышев» (60 TFlops) и «Ломоносов» (1.7 PFlops) [5-8], а также на перспективных высокопроизводительных системах, планируемых к установке в МГУ. Основные задачи проектируемого программного комплекса сформулированы следующим образом:

- обеспечение максимальной сохранности оборудования вычислительного комплекса;
- обеспечение максимально полного использования оборудования;
- поддержка преемственности опыта сопровождения суперкомпьютерных систем.

В основе проекта лежит идея построения модели функционирования современного суперкомпьютерного комплекса, отражающей основные компоненты суперкомпьютеров и их взаимосвязь, ориентированной на повышение продуктивности работы суперкомпьютерных систем, на оперативное обнаружение и локализацию аварийных ситуаций. Наличие адекватной модели позволяет создать систему, гарантированно контролирующую все аспекты работы комплекса. Модель функционирования суперкомпьютерного комплекса позволит оценить степень влияния аварийной ситуации на смежные блоки и подсистемы суперкомпьютера, минимизировать объем автоматически отключаемого оборудования и поддерживать максимально возможную часть нормально функционирующего оборудования в работоспособном состоянии.

Исходя из многолетнего опыта коллектива разработчиков по проектированию и сопровождению суперкомпьютерных систем [9-10], к разрабатываемой системе были установлены следующие требования:

- обнаружение всех потенциальных источников отказов комплекса. Помимо очевидных причин отказов и аварий, которые могут быть определены исходя из структуры модели, создается банк нетривиальных аварийных ситуаций, в котором аккумулируется реальный опыт поддержки больших вычислительных комплексов. Такой банк, в частности, упростит верификацию модели.
- наличие средств реагирования на аварийные ситуации, включающие как их автоматическое устранение (отключение или перезагрузка оборудования, перезапуск программных компонентов), так и оповещение персонала.
- гибкость и независимость от целевой архитектуры. Система должна быть настраиваемой на работу с суперкомпьютерным комплексом, построенным на основе наиболее распространенных программно-аппаратных архитектур.
- масштабируемость. Система должна как поддерживать суперкомпьютеры текущего поколения тера- и петафлопсной производительности, так и быть готовой к работе с вычислительными комплексами следующего поколения, в которых число компонентов будет увеличено на порядки.
- наличие средств самодиагностики. На практике регулярно возникают ситуации, когда имеющиеся системы мониторинга и обеспечения отказоустойчивости не предупреждают об аварийных ситуациях из-за отказа датчиков, проблем с собственным функционированием и т.д. Очевидно, что механизмы самодиагностики должны быть заложены внутри разрабатываемой системы. При этом они также должны опираться на модель суперкомпьютерного комплекса, контролируя корректность работы всех доступных средств мониторинга и оповещения.
- обеспечение преемственности в сопровождении суперкомпьютерных комплексов. Это требование подразумевает учет накопленного опыта, выраженного в нашем случае в виде связки модели комплекса и банка аварийных ситуаций, при установке на новые вычислительные системы.

- наличие удобного интерфейса для работы с системой, в т.ч. средств визуализации отдельных компонентов, логических и физических блоков суперкомпьютерного комплекса.

Модель суперкомпьютерного комплекса

В основу представления модели функционирования суперкомпьютерного комплекса положена структура мультиграфа.

Вершины мультиграфа, или объекты, соответствуют всем программным или аппаратным компонентам суперкомпьютера, которые необходимо контролировать (горячие коридоры, стойки, ИБП, процессоры, процессорные ядра и т.д.). Каждому объекту приписывается набор атрибутов – некоторых характеристик объекта, которые будет отслеживать создаваемая система. Все атрибуты разделяются на независимые и вычисляемые. Значения независимых атрибутов определяется только значениями датчиков – внешних источников информации о состоянии и поведении компонентов суперкомпьютера. В качестве источника данных будет выступать, в частности, установленная на суперкомпьютере система (или системы) мониторинга. Значения вычисляемых атрибутов формируются на основе других атрибутов после вызова определенных функций. К примеру, температура процессора вычислительного узла является независимым атрибутом, а свойство загруженности узла, определяемое на основе текущей загрузки всех ядер его процессора, будет вычисляемым атрибутом.

Ребра мультиграфа соответствуют физическим или логическим связям между соответствующими компонентами суперкомпьютера. Это могут быть связи типа «содержит» (шасси_1 содержит вычислительный_узел_1,..., вычислительный_узел_N), «питает» (ИБП_1 обеспечивает питанием стойку_1), «охлаждает» (кондиционер_1,.., кондиционер_N отводят тепло из горячего_коридора_1), «Infiniband», «Ethernet» (узел_1,..., узел_N соединены с коммутатором_1). Допускается создание связей любого типа.

В качестве хранилища и представления графовой модели вычислительной системы выбрана open-source графовая СУБД Neo4j [11], реализованная на языке Java. Разработан API для построения модели вычислительного комплекса, содержащей основные типы объектов и связей между ними. С использованием этого API в программе на Java генерируется модель конкретной вычислительной системы.

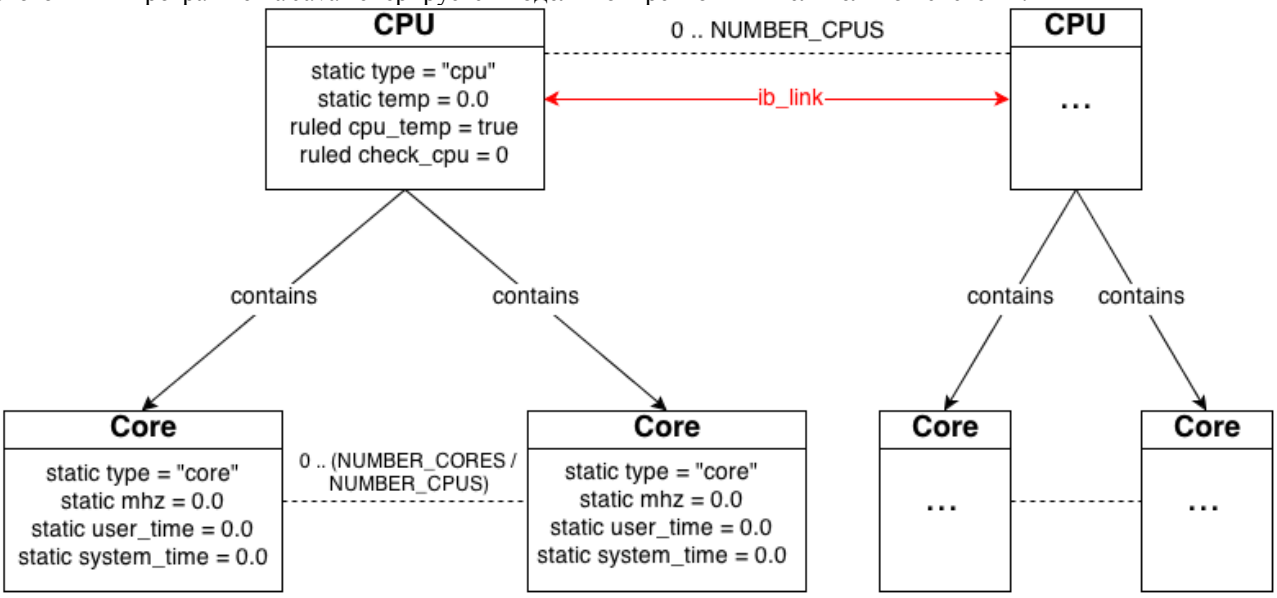

Рис. 1. Тестовая модель вычислительной системы в виде графа

Рассмотрим простую тестовую модель вычислительной системы, содержащую объекты типа «процессор» с вычисляемыми атрибутами «температура процессора» и «ошибки процессора» и объекты «процессорные ядра» с независимыми атрибутами «тактовая частота», «user_time» и «system_time», при этом между процессорами и заданным числом ядер установлена связь типа «содержит», а между процессорами — связь типа «Infiniband» (рис. 1). На рис. 2 приведено описание этой модели на языке на Java с помощью разработанного API. Опишем по строкам процесс создания модели:

1: Создание пустой модели;

3-4: Объявление функций контроля температуры и проверки ошибок ЦПУ. Пока эти функции не привязаны ни к одной вершине графа;

6-15: Объявление атрибутов, которые пока также не привязаны к вершинам графа;

17-20: Создание вершин графа, соответствующих процессорам и ядрам реальной системы. Также одновременно выполняется привязка атрибутов к данным вершинам;

22-25: Описание типов связей, которые будут использоваться в данной модели;

27: Создание связей типа «содержит» между процессорами и соответствующими ядрами;

28: Создание связей типа «сеть Infiniband» между всеми процессорами (топология «все-ко-всем»).

Описание модели вычислительного комплекса в виде программы на языке Java не всегда является удобным. Поэтому параллельно разрабатывается язык описания моделей. В качестве требований к разрабатываемому языку предъявляются простота и удобство описания, легкая читаемость и возможность генерации внешними автоматическими средствами. На рис. 3 приведен фрагмент программы на данном языке, создающий модель, аналогичную описанной выше. Описание модели на этом языке транслируется в Java-программу. Язык содержит конструкции для создания объектов и связей между ними. Имеется возможность подгружать данные из внешних файлов в формате CSV. Например, если имеется CSV-файл, содержащий список имен и сетевых адресов узлов конкретного суперкомпьютера, то одной языковой конструкцией можно сопоставить данные из этого файла и значения соответствующих атрибутов объектов типа «узел» модели, тем самым «привязав» модель к реальной вычислительной системе.

```
graph = new CNeo4jGraph("dbs/graphit_10", CNeo4jGraph.Op.RECREATE);
graph.GetIndex().EnableObjectIndex("type");
IAutoRule cpu_temp = new CUpperThreshold("check_cpu_temp", "CPU", "temp", 50.0);
IAutoRule check_cpu = new CCheckBoolRules("cpu_errors", cpu_temp);

IAttribute[] cpu_attributes = new IAttribute[]{
        new CStaticAttribute("type", "cpu"),
        new CStaticAttribute("temp", 0.0),
        new CRuledAttribute(cpu_temp, true),
        new CRuledAttribute(check_cpu, 0) };
IAttribute[] core_attributes = new IAttribute[]{
        new CStaticAttribute("type", "core"),
        new CStaticAttribute("mhz", 0.0),
        new CStaticAttribute("user_time", 0.0),
        new CStaticAttribute("system_time", 0.0) };

IAutoList<IAutoObject> cpu = new CNodeFactory(graph, true, cpu_attributes)
        .Create(NUMBER_CPUS);
IAutoList<IAutoObject> cores = new CNodeFactory(graph, true, core_attributes)
        .Create(NUMBER_CORES);

CRelationshipConnectorFactory contains = new CRelationshipConnectorFactory
        (graph, new CStaticAttribute("type", "contain"));
CRelationshipConnectorFactory ib_link = new CRelationshipConnectorFactory
        (graph,  new CStaticAttribute("type", "ib"));

contains.EveryToChunks(cpu, cores);
ib_link.AllToAll(cpu, cpu);
```

Рис. 2. Пример описания фрагмента модели тестовой вычислительной системы на языке Java

```
1  rule CUpperThreshold cpu_temp ("check_cpu_temp", "CPU", "temp", 50.0);
2  rule CCheckBoolRules check_cpu ("cpu_errors", cpu_temp);
3
4  group cpu_attributes {
5          static ("type", "cpu"),
6          static ("temp", 0.0),
7          ruled (cpu_temp, true),
8          ruled (check_cpu, 0) };
9
10 group core_attributes {
11         static ("type", "core"),
12         static ("mhz", 0.0),
13         static ("user_time", 0.0),
14         static ("system_time", 0.0) };
15
16 list[NUMBER_CPUS] cpu = object(cpu_attributes);
17 list[NUMBER_CORES] cores = object(core_attributes);
18
19 link_type contains;
20 link_type ib_link;
21
22 contains.EveryToChunks(cpu, cores);
23 ib_link.AllToAll(cpu, cpu);
```

Рис. 3. Пример описания тестовой модели вычислительной системы на специальном языке

Функции, соответствующие вычисляемым атрибутам, реализуются в виде классов Java. На рис. 4 приведен пример функции, которая проверяет, что количество кондиционеров с допустимой температурой не ниже определенного порога. Результат работы функции – булево выражение.

```
1  public class CAESSTempRule extends CBaseRule
2  {
3          int lower, upper, num_out;
4
5          @Override
6          public EDependencyType GetDeps()
7          {
8                  return EDependencyType.IN;
9          }
10
11         @Override
12         public Object Compute(IAutoObject object)
13                 throws ExceptionModelFail
14         {
15                 int count
16                         = object.GetInNeighbors("type", "chill")
17                                 .GetAttr("ColdTemp").ge(upper).size()
18                         + object.GetInNeighbors("type", "chill")
19                                 .GetAttr("ColdTemp").le(lower).size();
20
21                 return count >= num_out;
22         }
23 }
```

Рис. 4. Пример функции вычисления атрибутов

Рассмотрим по строкам процесс задания функции:
3: Объявление константных порогов для функции;
6-9: В данных строках задается, что данная функция зависит от соседних объектов, имеющих входящие ребра;
12: Начало объявления метода для вычисления значения функции;
15-19: Вычисление числа кондиционеров с допустимой температурой не ниже определенного порога;

21: Проверка полученного значения на соответствие порогу.

Основные принципы работы системы контроля суперкомпьютерного комплекса

Построенную модель вычислительной системы необходимо «оживить» – наполнить ее реальными данными, поступающими от систем мониторинга суперкомпьютерного комплекса, т. е. сопоставить независимым атрибутам актуальные значения датчиков. Это обеспечивается разрабатываемой системой контроля суперкомпьютерного комплекса. Планируемая архитектура прототипа системы приведена на рис. 5.

«Модуль Обработки Входных Данных» собирает значения датчиков мониторинга и передает их модулю «Логика Работы с Данными». Тот определяет, какие данные изменились и каким именно атрибутам в графе они должны соответствовать, после чего отдает их модулю «Графовая Модель Суперкомпьютера». При каждом изменении значений независимых атрибутов происходит вычисление всех связанных с ними вычисляемых атрибутов. За это отвечает модуль «Логика Работы с Функциями», а методы вычисления хранятся в модуле «Хранилище Функций». Далее модуль «Логика Работы Правилами» производит проверку выполнения правил – условий, накладываемых на состояние графа, касающихся в основном значений атрибутов. Правила хранятся в модуле «Хранилище Правил». В рамках правила может быть задана проверка корректного с точки зрения системы значения атрибута, в частности, его нахождение в заданных пределах. Если правило срабатывает (например, температура горячего коридора вышла за пределы допустимого порога), оно вызывает один из механизмов реакции модуля «Обработчик Реакций». Реакцией может быть оповещение ответственных лиц путем отправки SMS или e-mail, отключение питания компонентов вычислительной системы, перезагрузка вычислительных узлов.

Правила могут срабатывать не только по изменению значений атрибутов, но и по таймеру. Это необходимо для выявления ситуаций, когда неисправным может оказаться сам датчик, при этом предоставляемое им значение укладывается в заданные допуски, но со временем не изменяется.

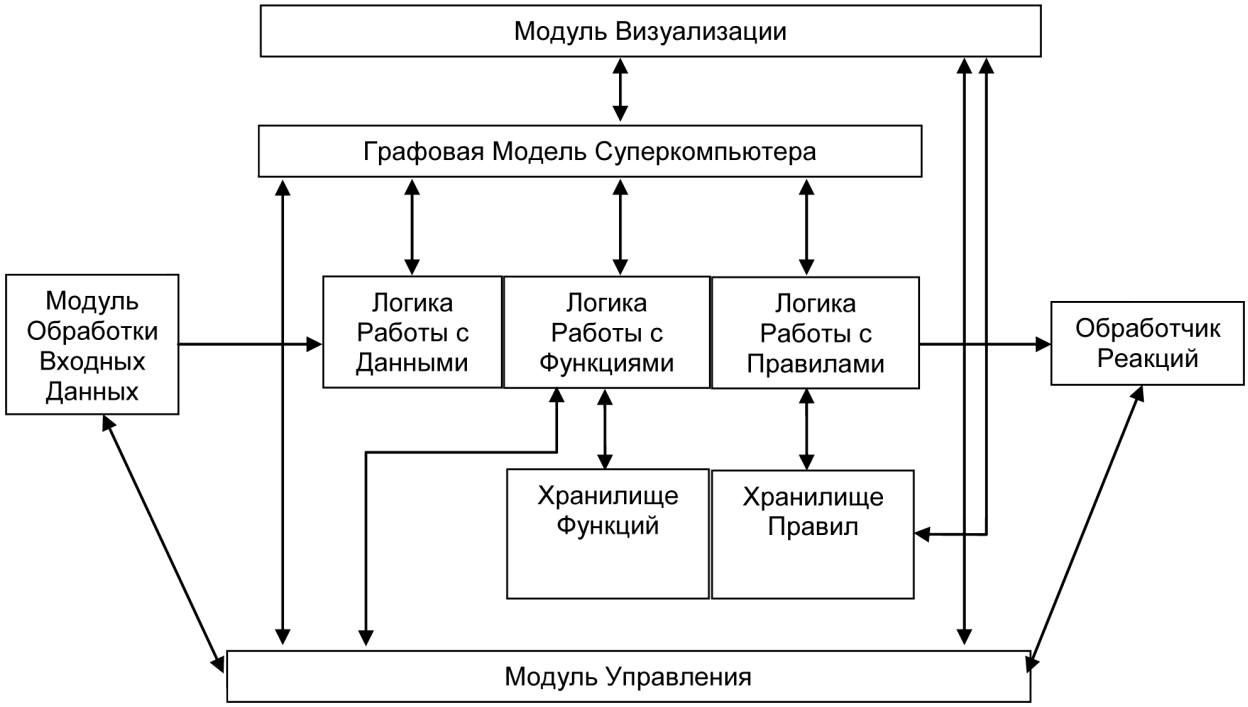

Рис. 5. Архитектура прототипа программного комплекса

Готовые наборы вычисляемых атрибутов и правил лягут в основу банка неисправностей и аварийных ситуаций, независимых от конкретной архитектуры вычислительного комплекса. Базовый набор таких атрибутов и правил можно использовать для адаптации системы к конкретному комплексу, при этом их можно свободно дополнять и модифицировать. Этим будет обеспечиваться требование преемственности.

Для реализации самодиагностики в системе будет предусмотрен механизм внутренних тестов, прохождение которых будет проверяться внешними сервисами. Внутренние тесты будут основаны на тестовых датчиках, не соответствующих реальным объектам вычислительной системы, но изменяющимся по определенным законам. Набор тестов должен включать в себя:

- проверку гарантированно рабочей цепочки атрибутов (данные от тестового датчика должны меняться, чтобы происходил пересчет значений атрибутов);
- проверку цепочки атрибутов, конечный датчик которой не обновляется;
- проверку цепочки атрибутов, конечный датчик которой никогда не отдает значения;

- проверку цепочки атрибутов, конечный датчик которой периодически становится недоступным;
- проверку запуска действия по заданному значению датчика;
- проверку отработки некорректных значений датчика;
- замер времени работы рабочего цикла программы.

В качестве расширенных средств самодиагностики может быть организован независимый сбор статистики поступления информации от датчиков и анализ динамики их изменения.

Согласно проведенным тестам, графовая СУБД Neo4j и модели на ее основе обладают достаточной производительностью при размере графа, соответствующем размеру модели суперкомпьютера «Ломоносов». Для увеличения потенциала масштабируемости модель суперкомпьютерного комплекса может быть представлена как совокупность моделей, содержащих различный набор объектов и/или связей между ними. Каждая такая модель может обрабатываться независимо отдельными экземплярами системы. Более того, в случае нехватки производительности Neo4j может быть произведена миграция на другую технологическую платформу. Это потребует только реализацию нужных системных API, при этом описание модели (включая атрибуты и правила) не потребует никаких изменений.

Текущие результаты и заключение

Разработка системы находится в активной стадии, поэтому некоторые описанные решения могут претерпевать существенные изменения. В частности, обсуждается вопрос об объединении понятий вычисляемых атрибутов и правил. Пока не реализованы модули визуализации и управления. В качестве тестовых средств визуализации использовалась инструментарий Graphviz [12], однако с его помощью можно отрисовывать только элементарные модели. Уже сейчас ощущается нехватка средств для проверки корректности создаваемых моделей.

Нами предложен подход к описанию модели функционирования суперкомпьютерных комплексов в виде мультиграфа. Созданы программные интерфейсы для генерации моделей. Сгенерирован ряд моделей, включая модель суперкомпьютера «Чебышев». Созданы средства описания правил выявления аварийных ситуаций, с помощью которых, в частности может быть описана логика работы системы аварийного автоматического отключения суперкомпьютера «Ломоносов».

Сформирована архитектура прототипа системы обеспечения оперативного контроля и эффективной автономной работы суперкомпьютерных комплексов. В данный момент реализуется отображение реальных данных, получаемых от системы агрегации данных мониторинга LAPTA [13-18], разработанной в рамках совместного российско-европейского проекта HOPSA [19], на модель суперкомпьютера «Чебышев». К концу 2013 г. планируется запуск системы в тестовом режиме на суперкомпьютере «Чебышев».

Работа выполняется при финансовой поддержке РФФИ в рамках научных проектов №12-07-33047 мол_а_вед и №13-07-00750 А.

ЛИТЕРАТУРА:

1. Система Nagios, http://www.nagios.org/
2. Система Ganglia, http://ganglia.sourceforge.net/
3. Система Zabbix, http://www.zabbix.com/
4. Система Zenoss, http://www.zenoss.org/
5. Суперкомпьютерный комплекс МГУ, http://parallel.ru/cluster
6. Moscow University Supercomputing Center, http://hpc.msu.ru
7. Воеводин Вл.В., Жуматий С.А., Соболев С.И., Антонов А.С., Брызгалов П.А., Никитенко Д.А., Стефанов К.С., Воеводин Вад.В. Практика суперкомпьютера "Ломоносов" // Открытые системы. - М.: Издательский дом "Открытые системы", 2012. - 7: ISSN 1028-7493.
8. Sadovnichy V., Tikhonravov A., Voevodin Vl, Opanasenko V. “Lomonosov”: Supercomputing at Moscow State University. Contemporary High Performance Computing: From Petascale toward Exascale.- Chapman & Hall/CRC Computational Science, CRC Press.- Boca Raton, United States.- pp. 283-307
9. Воеводин Вл.В., Жуматий С.А. «Вычислительное дело и кластерные системы».-М.: Изд-во МГУ, 2007. - 150 с.
10. Воеводин Вл.В., Жуматий С.А. Экспонента суперкомпьютерных центров.// Открытые системы. - М.: Издательский дом "Открытые системы", 2008.- 5.- с. 12–15
11. Система Neo4j, http://www.neo4j.org/
12. Система Graphviz, http://www.graphviz.org/
13. Никитенко Д.А., Стефанов К.С. Исследование эффективности параллельных программ по данным мониторинга // Вычислительные методы и программирование.- 2012.- Т. 13.- Раздел 2. С. 97-102 (http://num-meth.srcc.msu.ru/).
14. Адинец А.В., Брызгалов П.А., Воеводин Вад.В., Жуматий С.А., Никитенко Д.А., Стефанов К.С. HOPLANG: развитие языка обработки потоков данных мониторинга // Вычислительные методы и программирование.- 2012.- Т. 13.- Раздел 2. С. 126-131 (http://num-meth.srcc.msu.ru/).

15. Adinets A.V., Bryzgalov P.A., Voevodin Vad.V., Zhumatii S.A., Nikitenko D.A., Stefanov K.S. Job Digest: an approach to dynamic analysis of job characteristics on supercomputers // Numerical methods and programming: Advanced Computing.- 2012.- V. 13.- Section 2. Pages. 160-166 (http://num-meth.srcc.msu.ru/).
16. Адинец А.В., Брызгалов П.А., Воеводин В.В., Жуматий С.А., Никитенко Д.А., Стефанов К.С. JobDigest - подход к исследованию динамических свойств задач на суперкомпьютерных системах // Вестник Уфимского государственного авиационного технического университета.- 2013.- Т. 17.- № 2(55).- С. 131-137
17. Брызгалов П.А., Жуматий С.А., Никитенко Д.А., Адинец А.В. Система визуализации параметров работы больших вычислительных систем // Параллельные вычислительные технологии 2012 (ПаВТ'12): Сборник трудов международной научной конференции.- 2012.- с. 714
18. Адинец А.В, Брызгалов П.А, Воеводин Вад.В, Жуматий С.А, Никитенко Д.А. Мониторинг, анализ и визуализация потока заданий на кластерной системе// Высокопроизводительные параллельные вычисления на кластерных системах: Материалы XI Всероссийской конференции (Нижний Новгород, 1–3 ноября 2011 г.).- Нижний Новгород: Издательство Нижегородского госуниверситета.- 2011.- с. 10-14
19. Bernd Mohr, Vladimir Voevodin, Judit Giménez, Erik Hagersten, Andreas Knüpfer, Dmitry A. Nikitenko, Mats Nilsson, Harald Servat, Aamer Shah, Felix Wolf, and Ilya Zhujov. The HOPSA Workflow and Tools. Proceedings of the 6th International Parallel Tools Workshop, Stuttgart, September 2012, Springer (to appear).