Трёхуровневая система мониторинга расширенной функциональности

А.Г. Тарасов

В статье рассматривается трёхуровневая архитектура системы мониторинга в применении к обслуживанию вычислительных ресурсов, иерархическая структура представления, хранения и организации передачи данных, применение базовых концепций архитектуры к расширению функциональности существующих систем, триггеры событий и простейшие механизмы саморегуляции вычислительной системы. Затронуты вопросы особенностей практической реализации на языке java. Представлены сравнительные результаты тестирования с системой Ganglia на кластере ВЦ ДВО РАН.

1. Введение

Задачей системы мониторинга является своевременное предоставление актуальной информации оператору вычислительной системы, как в целом так и по отдельным узлам контролируемого комплекса. Простой и оперативный доступ к этой информации позволяет адекватно реагировать на все негативные изменения в работе системы, находить причины неисправностей и сбоев. Система мониторинга может являться средством отладки и наблюдения за поведением выполняющихся пользовательских программ, позволяя в реальном времени наблюдать расходуемые программой ресурсы и выявлять в ней узкие места.

При выборе системы мониторинга для высокопроизводительных систем необходимо учитывать специфику большинства параллельных приложений, выполняемых на этих системах. Такие приложения оказывают наиболее сильную нагрузку на центральный процессор и оперативную память. Время выполнения программ нередко составляет несколько недель. Поэтому необходимыми требованиями к системе мониторинга является минимально возможное использование ресурсов системы (центральный процессор, оперативная память, подсистема хранения данных), возможность удаленной визуализации данных и наблюдения за происходящими в вычислительной системе событиями.

Специализированные высокопроизводительные системы зачастую обладают аппаратными средствами, выполняющими функциями мониторинга (например, сервисные платы). Однако, для вычислительных систем, построенных по принципам Beowulf-кластеров, приходится использовать в работе либо неспециализированные системы (например, Nagios [1]), либо созданные для мониторинга кластеров (широко используемая система Ganglia [2]). Расширяемость подобных систем в плане удобства добавления необходимой функциональности обычно мала, базовый набор сервисов невелик.

В частности, в системе Ganglia отсутствует возможность использования механизма триггеров и уведомлений о событиях. Обладая хорошей масштабируемостью Ganglia, в то же время, не позволяет организовывать структуры передачи данных, отличные от иерархических. Nagios изначально создавалась для мониторинга сетевых сервисов и позволяет добавлять модули, разработанные сторонними разработчиками, однако добавление сервиса в общем случае требует остановки системы.

В данной статье рассматривается подход, позволяющий использовать указанные системы в рамках трёхуровневой архитектуры организации мониторинга с возможностью внесения улучшений, не нарушая при этом работоспособности систем.

2. Трёхуровневая архитектура

Терминология, используемая ниже при рассмотрении архитектурных особенностей обсуждаемого подхода, в некоторой степени пересекается с понятиями широко используемых систем мониторинга. В то же время, ряд терминов обозначает отличные логические сущности.

Трёхуровневая система мониторинга [3] строго разделяет весь использующийся при мониторинге программно-аппаратный комплекс на три уровня. Передача данных может осуществляться лишь между соседними уровнями, ослабляя тем самым зависимости между крайними уровнями и позволяя изменять, дополнять и расширять уровни не нарушая общей работоспособности.

Нижний уровень ответственен за сбор и представление данных в виде *метрик*. Источником данных может быть какой-либо *сенсор* физического или логического устройства (например SNMP-статистика, данные сервиса gmond). Функциональность уровня мала, в целях более эффективного использования ресурсов контролируемого узла ПО на практике – машиннозависимое.

Промежуточный уровень отвечает за сбор данных с нескольких источников данных, преобразование их к унифицированному формату данных, проверку простейших *триггеров* и выполнение соответствующих им *действий*. Данный уровень можно реализовать на машиннозависимых или интерпретируемых языках.

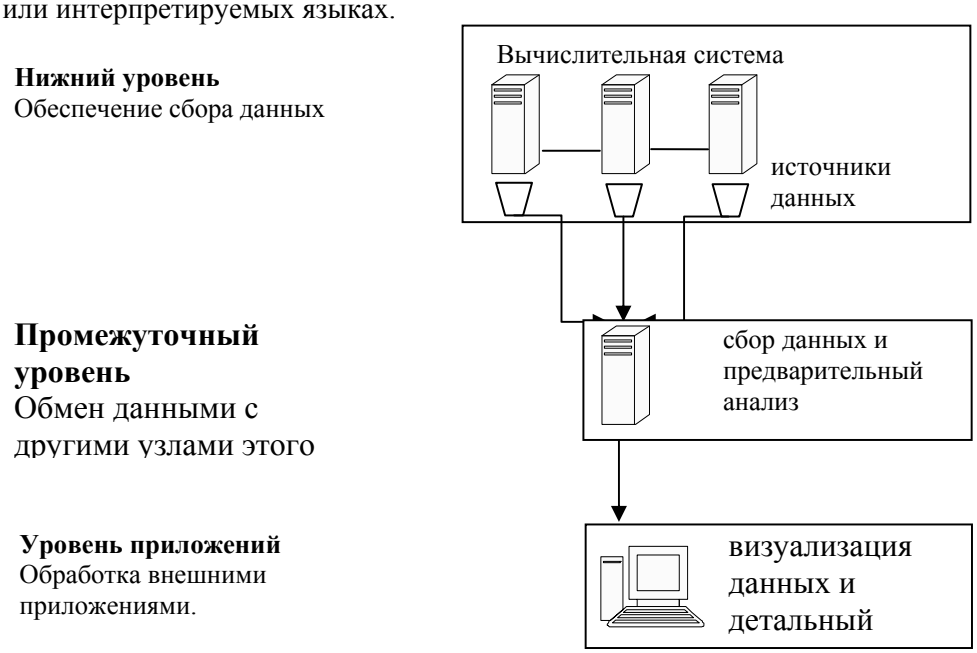

Рис. 1. Три уровня системы мониторинга

Уровень приложений отвечает за более высокоуровневый анализ данных и представление их пользователю. Быстродействие на данном уровне не так важно, т. к. выполняющееся на нём программное обеспечение может выполняться за пределами вычислительного комплекса, обращаясь лишь к данным, предоставляемым промежуточным уровнем.

Базовой единицей хранения данных является метрика. Метрика – это набор данных определённого типа (в общем случае тип данных для всех хранимых значений неодинаков, однако, подобные метрики применяются редко).

Метрики бывают неизменяемые со временем (или *статические)* и изменяемые (т. е. *динамические)*. Кроме того метрики делятся на *длительные* (обеспечивают доступ ко всем данным измерений) и *моментальные* (хранят только текущее значение). Программный комплекс, реализующий архитектуру должен отслеживать непротиворечивость данных по времени, т. е. соблюдение порядка помещения данных в метрику. На практике метрика хранит лишь набор данных за относительно небольшой промежуток времени и организует эффективный доступ к ним. К остальным данным, выгружаемым из оперативной памяти во внешнее хранилище данных, прозрачно для пользователя обеспечивается доступ по запросу.

Метрика существует только в рамках *узла* контролируемой системы. Следует отличать вычислительный узел (host, в терминологии Ganglia) от узла иерархической структуры данных (node). Далее в тексте под узлом понимается второе значение. Узел обладает изменяемыми во времени атрибутами и набором метрик.

Атрибут всегда хранит только одно значение, для которого явно не указан тип (более точ-

но, атрибут – это статическая моментальная метрика строкового типа). Атрибутами являются вспомогательные данные, используемые при работе с системой (например, атрибутом могут являться параметры командной строки, переданные приложению пользователя).

Узел (кроме корня иерархической структуры) всегда является дочерним по отношению к какому-либо другому узлу и может быть родительским по отношению к набору некоторых узлов. Архитектура предполагает динамическое формирование иерархической структуры, узлом которой может выступать любая определяемая *источником данных* (или - *сенсором*) сущность. Таким образом, применительно к мониторингу вычислительных кластеров могут существовать метрики гридов, кластеров, вычислительных узлов и задач. Естественным образом можно объединить эти узлы в иерархию, используемую системой Ganglia. Однако, возможно и любое другое объединение. Если на промежуточном уровне будет выполняться несколько модулей сбора данных, каждый из них может полагаться источником данных и таким образом являться узлом иерархии.

Каждый источник данных может производить измерение множества метрик, в общем случае неодинаковое для различных источников. Через определённые промежутки времени, источник данных делает мгновенный снимок состояния метрики и предаёт данные в процессе *сеанса* обмена данными на управляющий сервер, ответственный за обработку информации. Источник данных работает на нижнем уровне архитектуры, управляющий сервер на промежуточном или уровне приложений.

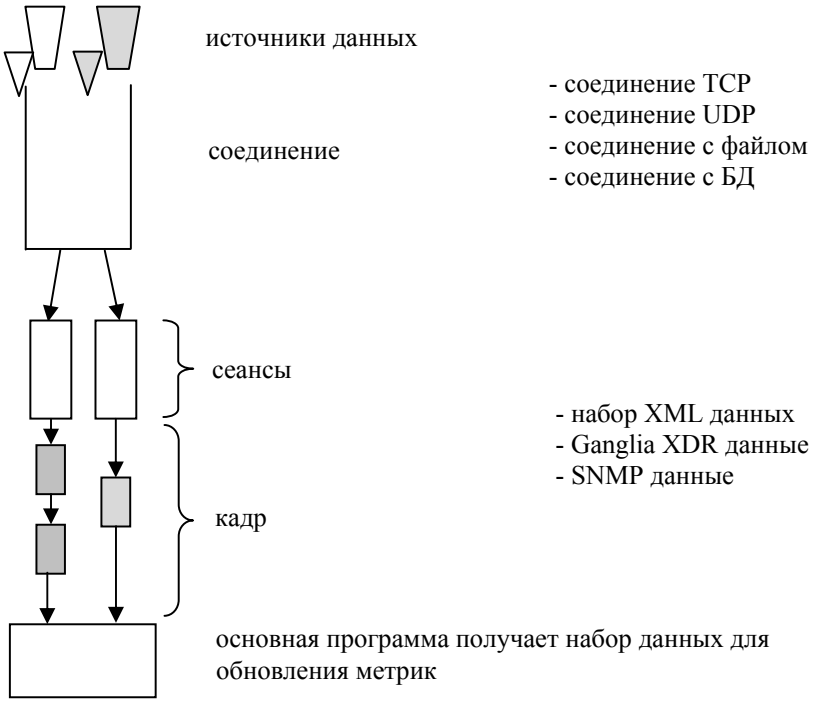

Рис. 2. Процесс получения данных метрик

Таким образом, для сенсора метрика существует лишь в момент времени. При следующем измерении конкретная метрика может уже не существовать или же наоборот, могут добавиться новые метрики. Момент замера фиксируется на собирающем узле.

Соединение - это инструмент, посредством которого обеспечивается физический доступ к данным. Результатом работы соединения является возможность открытия сеансов. Соединение следует трактовать в широком смысле – в зависимости от источника данных это может быть соединение с локальным файлом данных, с удалённым SNMP-сервером, с широковещательным UDP каналом, базой данных и т. д.

Соединение устанавливается к источнику данных единожды, при инициализации. В дальнейшем, приложение инициирует открытие сеанса получения данных, используя ранее настроенное соединение. Работа с сеансом может привести к созданию последовательности нескольких кадров.

Кадр – это набор данных, характеризующий метрики и их значения на определённый момент времени (т. е. механизм логического доступа к данным). В одном кадре содержатся данные для ноля и более метрик. Следует отметить, что хотя получаемые значения и дискретны по времени, обсуждаемая архитектура полагает значение метрики неизменным до получения следующего значения. Таким образом для ситуации, изображённой на рисунке 3, значение некоторой метрики на промежутке $[t_1, t_2)$ будет полагаться равным m_1 .

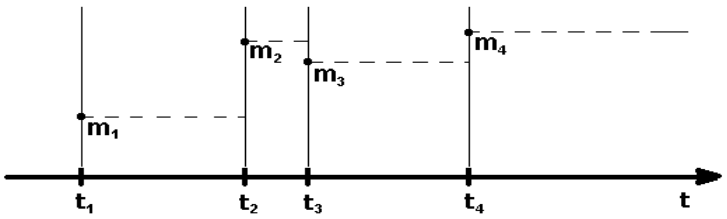

Рис. 3. Область применения значения метрики

Следующими важными структурными элементами архитектуры являются *триггер* и *действие*.

Триггер – это сущность, выполняющая проверку некоторого условия. Если условие выполнено – состояние триггера изменяется (триггер сработал), что приводит к исполнению некоторых *действий*. Триггеры и действия имеет смысл использовать совместно. При этом одному триггеру может быть сопоставлено несколько действий и наоборот, одному действию может быть сопоставлено несколько триггеров.

Триггеры делятся на *переключаемые* и *сбрасываемые.* Первые находятся в сработавшем состоянии до тех пор, пока условие выполняется, и не производят действий повторно. Вторые - при каждом срабатывании выполняют действие.

Действие – набор инструкций, реакция на срабатывание триггера. При необходимости, действия могут запускать на выполнение другие действия. Действие может широко варьироваться: запись диагностического сообщения в журнал, посылка уведомления администратору кластера, запуск приложений и др.

Генератор триггеров (или *метатриггер*) – триггер, создающий триггеры для метрик, подходящих под условия. Необходимость существования подобной сущности продиктована динамической структурой иерархии узлов системы мониторинга. При этом часть узлов уже может быть создана (и иметь активные триггеры), другая - отсутствовать и подключаться по необходимости. Метатриггеры облегчают добавление новых узлов в вычислительную систему.

Введение таких логических сущностей и программная реализация строго определённых интерфейсов взаимодействия между ними позволяют использовать совместно на различных уровнях программное обеспечение независимых разработчиков. Например, сенсорами могут выступать файлы данных, данные gmond. Однако, все сенсоры формируют единую структуру метрик, единообразно доступную программным средствам на промежуточном уровне и уровне приложений.

Таким образом, построенная в соответствии с данной архитектурой система мониторинга может работать параллельно с уже развёрнутыми, замещая их на некоторых уровнях, что позволяет изменять и расширять набор доступных функций этих систем.

3. Практическая реализация

Использование связки триггер-действие позволяет реализовать сколь угодно сложную схему обработки критических и штатных ситуаций, возникающих в процессе работы вычислительной системы (при условии возможности добавления своих действий и триггеров в систему мониторинга). В частности, третий уровень рассматриваемой архитектуры может использовать срабатывание триггеров как входные параметры для экспертной системы, принимающей решение о необходимости последующего воздействия на вычислительный процесс.

На экспериментальном кластере ВЦ ДВО РАН [4] система мониторинга grate, разрабо-

танная по рассматриваемой архитектуре, использовалась для контроля негативных изменений повышения температуры вычислительных узлов (например, в случае отказа активного охлаждения). Сенсорами выступали сервисы gmond, предоставляемые системой Ganglia, с предварительно настроенной пользовательской метрикой температуры. При превышении заранее указанной величины происходила запись в журнал событий, отправка уведомления администратору кластера (в критическом случае принудительное отключение узла выполнялось аппаратно, хотя подобная функциональность достижима и в рамках системы мониторинга).

Уровень приложений может использоваться и для более сложных управляющих операций над контролируемой системой. В качестве примера можно привести оптимизацию производительности гетерогенной вычислительной системы.

В ряде случаев приложения пользователей кластера занимают не все вычислительные ресурсы узла, на котором выполняются, в то время как более ёмкие с точки зрения необходимых ресурсов памяти или процессорного времени задачи вынуждены простаивать в очередях в ожидании необходимых узлов. При использовании механизма виртуализации XEN на кластере ВЦ ДВО РАН [5] была добавлена возможность миграции (с некоторыми ограничениями) выполняющихся процессов с одного вычислительного узла на другой.

Системы виртуализации позволяют запускать несколько копий операционных систем на одной машине. При этом одна из операционных систем выполняет служебные функции и называется хостовой (от англ. host), а остальные – гостевой. Миграция выполняющейся копии операционной системы доступна с использованием стандартных средств XEN. Дополнительная возможность создания слепка выполняющейся операционной системы (т.н. checkpoint) может использоваться для устранения нарушения работоспособности при восстановимом сбое.

Программный комплекс анализировал эффективность использования ресурсов на вычислительном узле, используя предоставляемые промежуточным уровнем данные (эффективная загрузка процессора, объём занятой памяти). И при необходимости осуществлял миграцию с одного узла на другой.

Так же система использовалась при моделировании восстановимого отказа узла (т. е. такого, при котором миграция ещё возможна. Например, выход из строя несистемной среды распространения данных, перегрев со снижением частоты). Задачей системы было отследить отказ оборудования и инициировать миграцию, выполнив заранее подготовленный shell-скрипт.

При этом не происходило значительного прерывания вычислительного процесса. Как показали тестовые замеры, для малого числа узлов, участвующих в расчёте параллельной задачи с постоянной передачей данных между узлами, потери были малы [6].

Разработанный программный комплекс grate в дополнение к перечисленным в предыдущем разделе сущностям включает в себя область вывода данных, в которой осуществляется представление данных (таблица, график и т.п.).

grate состоит из нескольких самодостаточных модулей, написанных на языке java: grated (аналог gmond, способный замещать его на промежуточном уровне), grape (java-апплет, выполняющийся в среде web-браузера), grate (приложение для простейшей визуализации и контроля данных). Модули используют общее ядро (систему интерфейсов и классов), позволяющее им работать с метриками, триггерами и прочими объектами.

Выполненное при мониторинге 9 узлов тестирование экспериментального кластера, а затем при моделировании контроля 100 узлов по методике, использующейся разработчиками Ganglia - показало что производительность комплекса (модуля grated) чуть выше чем у системы Ganglia 2.4 (модуля gmetad), уступая ей в объёмах используемой памяти на управляющем сервере [7]. Модуль grated проводил проверку триггеров, расширяя тем самым базовую функциональность используемых сенсоров gmond. Визуализация выполнялась grate на основе данных, поставляемых от grated.

Тестирование проводилось на сервере локальной вычислительной сети кластера ВЦ ДВО РАН под управлением WhiteBox Linux 3.0 2.4.20-8. Результаты приведены в таблице, где показаны значения общего времени выполнения T_0 и относительное время загрузки процессора T_p. Использовалась Sun JRE 1.5.

Таблица 1. Результаты испытаний системы мониторинга

Выполняемое приложение	Кластер, 8 узлов		Эмуляция кластера, 100 узлов	
	T_0 (час.)	$T_р$ %	T_0 (час.)	$T_р$ %
grated	768	0.14	50.8	9.19
gmetad	888	0.23	192	10.6

Использование языка java позволило сделать систему гибкой и легко расширяемой: для добавления нового триггера или действия достаточно реализовать интерфейс, определённый в ядре системы. Созданный таким образом модуль можно начинать использовать не останавливая работы системы благодаря механизму отражений (reflection), поддерживаемому языком java.

Также стала возможной совместная работа приложений на разных платформах (в тестировании участвовали ОС MS Windows XP и различные версии дистрибутивов Linux)

4. Заключение

Разработанное по предложенной архитектуре приложение показало сравнимые результаты с уже имеющимися решениями, расширив функциональность используемой на нижнем уровне системы мониторинга (её источников данных). Таким образом на практике была показана работоспособность архитектуры и выгода от её применения.

Использование программных комплексов, реализующих подобную архитектуру, упрощает создание высокоуровневого программного обеспечения, контролирующего вычислительные системы. В частности, позволяет в ряде случаев использовать экспертные системы и другие методы из области моделирования искусственного интеллекта для управления вычислительной системой.

Дальнейшие исследования будут связаны с разработкой более «интеллектуального» программного обеспечения контроля вычислительного кластера ВЦ ДВО РАН на базе изложенной трёхуровневой архитектуры и применение его для объединенных в GRID кластеров. Использование языка java в широко распространённом пакете Globus Toolkit нивелирует имеющиеся недостатки системы grate, связанные с использованием значительного объёма памяти под JRE.

Литература

1. Wolfgang Barth. Nagios System and Network Monitoring. No Starch Press, 2006.
2. Matthew L. Massie, Brent N. Chun, David. E. Culler. The Ganglia Distributed Monitoring System: Design, Implementation, and Experience. UC Berkeley, 2003
3. Тарасов А. Г. Мониторинг вычислительного кластера с использованием java-технологий // XXX Дальневосточная математическая школа-семинар имени академика Е.В. Золотова: тезисы докладов. - Хабаровск : Изд-во ДВГУПС, ИПМ ДВО РАН, 2005, с. 201
4. Пересветов В. В., Сапронов А. Ю., Тарасов А.Г. Вычислительный кластер бездисковых рабочих станций // Препринт №83. - Хабаровск: Вычислительный центр ДВО РАН, 2005. - 50 с.
5. Пересветов В. В., Сапронов А. Ю., Тарасов А.Г., Шаповалов Т.С. Организация работы вычислительного кластера в режиме удалённого доступа // Препринт №110. - Хабаровск: Вычислительный центр ДВО РАН, 2007. - 34 с.
6. Сапронов А. Ю., Тарасов А. Г., Шаповалов Т.С. Применение системы виртуализации XEN на вычислительном кластере // XXXII Дальневосточная математическая школа-семинар имени академика Е.В. Золотова : тезисы докладов. - Хабаровск : Изд-во Дальнаука, ИПМ ДВО РАН, 2007. с. 174
7. Пересветов В. В., Сапронов А. Ю., Тарасов А. Г., Шаповалов Т. С. Удалённый доступ к вычислительному кластеру ВЦ ДВО РАН // Вычислительные технологии, т. 11 – Новосибирск: Изд-во ИВТ СО РАН, 2006, с. 45-51