УДК 681.3.06 + 519.68

РАСШИРЯЕМАЯ СИСТЕМА МОНИТОРИНГА ВЫЧИСЛИТЕЛЬНОГО КЛАСТЕРА

А. Г. Тарасов[1]

Рассматривается архитектура системы мониторинга вычислительных ресурсов. Предложены концепции расширения функциональности существующих систем с использованием триггеров событий и механизмов уведомлений. Разработанная система мониторинга реализована на языке программирования java. Затронуты вопросы ее применения в задачах управления вычислительным кластером, в том числе совместно с системой виртуализации XEN. Представлены результаты проведенных экспериментов с созданной системой мониторинга.

Ключевые слова: управление вычислительными кластерами, системы мониторинга, системы виртуализации.

1. Введение. При создании вычислительного комплекса необходимо решить ряд задач. Одной из них является развертывание системы мониторинга вычислительного комплекса с доступом к данным как в реальном времени, так и в режиме просмотра данных за определенный период, наблюдение за потреблением вычислительных и иных ресурсов как в целом, так и отдельными компонентами вычислительной системы.

В настоящей статье приведено описание архитектуры созданного кросс-платформенного программного комплекса grate, клиентская часть которого выполняет обработку и представление полученных данных. В качестве примера работы с источником данных использована система Ganglia 2.5.6, развернутая для мониторинга состояния вычислительного кластера ВЦ ДВО РАН. Рассматриваются также более сложные задачи, решаемые при помощи интеграции системы мониторинга в другие программные комплексы.

В тексте статьи под термином *кластер* понимается вычислительный кластер на базе неспециализированных компонентов (так называемый Beowulf-кластер), мониторинг которого решает задачи, являющиеся подмножеством задач контроля компьютерных систем в целом. Это позволило использовать в нашей работе некоторые идеи, реализованные для систем контроля сбоев и систем определения проникновения в локальную вычислительную сеть.

Специализированные высокопроизводительные системы часто обладают аппаратными средствами, выполняющими функции мониторинга, например сервисные платы. Полученные от них данные затем анализируются программным обеспечением, поставляющимся с системами, например IBM Tivoli Monitoring (http://www.ibm.com/developerworks/ru/tivoli/). Однако для вычислительных систем, построенных по принципам Beowulf-кластеров, обычно используются либо неспециализированные системы (например, система Nagios [1]), либо изначально созданные для мониторинга кластеров (например, широко используемая система Ganglia [2]). Расширяемость подобных систем в плане удобства добавления необходимой функциональности обычно мала, базовый набор сервисов невелик, и возможность взаимодействия различных систем мониторинга между собой отсутствует. Необходимость устранения этих ограничений являлось одной из причин создания новой системы мониторинга на основе синтеза нескольких подходов.

2. Построение системы мониторинга. В [3] описывается применение мониторинга для отслеживания попыток проникновения в контролируемую сеть, вводится понятие двух классов систем определения вторжения: сетевые (network-based) и узловые (host-based). Первые работают на основе контроля трафика в сети, вторые принимают решение на основе контроля данных с конкретных узлов. Описываемая в нашей работе система мониторинга относится к гибридной, поскольку большинство негативных процессов в вычислительных кластерах можно определить с использованием сенсоров, находящихся на конкретных узлах сети, однако в обсуждаемой реализации системы мониторинга grate агенты на узлах передают данные в сеть широковещательно и модуль сбора не обращается напрямую к узлам. В [4] такие специализированные системы мониторинга также делятся на централизованные, в которых сбор данных

[1] Вычислительный центр Дальневосточного отделения РАН (ВЦ ДВО РАН), ул. Ким-Ю-Чена, 65, 680063, г. Хабаровск; научный сотрудник, e-mail: taleks@as.khb.ru

© Научно-исследовательский вычислительный центр МГУ им. М.В. Ломоносова

производится в каком-либо одном месте, и распределенные, если не существует единственного центра сбора данных, а все узлы либо владеют полной информацией о системе, либо существует несколько узлов, ответственных за сбор информации. Предлагаемая в нашей статье система мониторинга является распределенной и иерархической и позволяет распределять в сети нагрузку, порожденную механизмами сбора и передачи необходимых данных. Кроме того, в [4] вводится понятие прямого (direct) и косвенного (indirect) мониторинга. К первому относится непосредственный доступ к данным контролируемого объекта, например к данным ядра операционной системы. Ко второму — доступ к тем же данным через посредника, которым может быть некоторый файл дампа.

Для определения состояния вычислительной системы применяются различные механизмы. В [5] предлагается распознавать класс состояния вычислительного кластера с помощью искусственных нейронных сетей. В [6, 7] обсуждается подход с использованием сетей Петри для диагностирования состояния контролируемой системы на основе уведомлений о событиях в системах с дискретными событиями.

Обработка событий и уведомление пользователей системы о них может осуществляться различными способами. Системами с уведомлением о событиях на базе подписки являются ISIS [8] и Gryphon [9]. Система XMLBlaster [10] использует язык XML для описания события и нахождения пути к адресату на базе информации о событии. Альтернативой такому подходу является уведомление всех пользователей системы обо всех событиях либо об уведомлении лишь администратора вычислительного комплекса. Подобный подход используется при мониторинге системной информации в операционных системах Linux при возникновении любых сообщений ядра.

Сложной задачей является опрос всех сенсоров и выработка удобного легковесного интерфейса для опроса всех известных сенсоров. В [11] приводится использование механизма SQL-запросов для получения и агрегирования необходимых данных из сети сенсоров. Одним из средств контроля состояния сетевых устройств является протокол SNMP (Simple Network Management Protocol), подробное описание которого приведено в RFC 1156 и RFC 1157 (Request For Comments — открытые стандарты в области развития и работы сетевых протоколов). Данный протокол может использоваться как средство доступа к аппаратным сенсорам ряда устройств. Несмотря на некоторые ограничения, SNMP-счетчики хорошо вписываются в архитектуры систем мониторинга как сенсоры нижнего уровня.

Мы обсуждаем в статье только прямой мониторинг на примере доступа к данным системы мониторинга Ganglia. Разработанный программный комплекс в описанных ниже экспериментах использует иерархическую схему сбора данных.

Основными характеристиками при мониторинге вычислительного кластера являются разделяемые между различными задачами ресурсы: количество доступных процессоров, объем используемой оперативной и дисковой памяти, нагрузка на сетевую подсистему, количество пользователей и т.п. Простой и оперативный доступ к этой информации позволяет оценить текущее состояние вычислительной системы, реагировать на негативные изменения в работе системы, находить причины неисправностей и сбоев, а также использовать полученную информацию в более высокоуровневых программных компонентах (диспетчеризация задач, системы виртуализации и др.).

Для пользователя система мониторинга является средством отладки и наблюдения за поведением конкретной задачи. Пользователь должен иметь возможность в реальном времени наблюдать расходуемые его программой ресурсы, выявлять в ней узкие места. При запуске параллельных программ пользователю обычно необходимо знать общий уровень загрузки кластера и доступные ресурсы кластера. При сбое работы приложения пользователь может использовать данные за некоторый период для анализа причин сбоя.

Как администратору, так и пользователю важно получать уведомления об изменениях в работе контролируемой системы, особенно критических для работы приложения или кластера. Необходима возможность гибкой настройки уведомлений различными способами: электронная почта, RSS-подписки и др.

При выборе системы мониторинга важно учитывать специфику параллельных приложений, выполняемых на вычислительных кластерах. Такие приложения оказывают наиболее сильную нагрузку на центральный процессор, оперативную память и, как правило, на коммуникационную сеть. Программа мониторинга должна минимально загружать центральный процессор, не использовать значительные объемы памяти и иметь возможность для удаленного наблюдения.

Программный комплекс системы мониторинга в нашей реализации представляет собой клиент-серверное приложение, состоящее из нескольких модулей, которые выполняются либо на вычислительных узлах (сенсоры), либо на выделенных узлах (ретрансляторы сообщений, центры сбора данных).

Данные поставляются сенсорами в центры сбора данных одним из доступных способов, например, широковещательным пакетом данных или ориентированным на соединение протоколом TCP (Transmis-

sion Control Protocol). Центры сбора данных выполняют предварительную обработку данных, их фильтрацию и генерирование сообщений о событиях.

Ретрансляторы сообщений совмещены с центрами сбора данных и предоставляют клиентским приложениям возможность подписываться на уведомления о событиях, снимая тем самым необходимость осуществления маршрутизации событий с вычислительных узлов. Любое приложение имеет возможность подписаться на уведомление о событиях, а также генерировать новые события. События обрабатываются в определенном порядке, зависящем от установленных пользователем фильтров событий.

Центры сбора данных могут обмениваться полученными данными между собой и выделенным главным центром сбора данных. Пользователь получает данные от такого центра посредством явного запроса с указанием необходимых параметров, если соответствующая реализация интерфейса отчета поддерживает их.

В целом программный комплекс разрабатывался таким образом, чтобы:

— иметь возможность уведомлять пользователя о событиях;

— быть легко настраиваемым для использования различных источников данных;

— корректно обрабатывать исключительные ситуации (задержка в получении данных, ошибка при получении данных, отсутствие данных и др.);

— предоставлять набор библиотек, интегрируемых в высокоуровневые программные комплексы;

— предоставлять изменяемый под необходимые нужды интерфейс пользователя;

— быть надежным и нетребовательным к ресурсам.

Поскольку вычислительный кластер не является географически распределенной вычислительной системой со свободным доступом из внешних сетей к вычислительным узлам, то при создании системы мониторинга предполагалось, что его безопасность будет обеспечиваться другими компонентами системного программного обеспечения. В большинстве практических реализаций вычислительных систем необходимые средства защиты информации (аутентификации клиентов, авторизации для доступа к ресурсам и т.д.) доступны посредством сервисов операционных систем или систем диспетчеризации задач.

Разработанный программный комплекс grate, исходный код стабильной версии которого передан в сетевой ресурс фонда открытого программного обеспечения sourceforge.net [12], представляет собой библиотеку классов языка java, обеспечивающую возможность выполнения базового набора функций по аналогии с системой Ganglia, что было сделано с целью облегчить совместное использование этих двух программных продуктов. Вместе с тем, разработанная архитектура является более универсальной, что позволяет использовать ее при минимальных изменениях и в других системах мониторинга, а также использовать различные системы мониторинга совместно.

Язык программирования java был выбран для работы над данным проектом, поскольку обладает следующими достоинствами: удобство использования стандартной библиотеки классов, надежность, платформенная независимость, возможность использования в web-среде.

Как показали результаты экспериментов, недостатки языка программирования java (невысокая производительность в некоторых задачах, значительный объем дополнительных библиотек) не оказывают существенного влияния на производительность в задачах мониторинга.

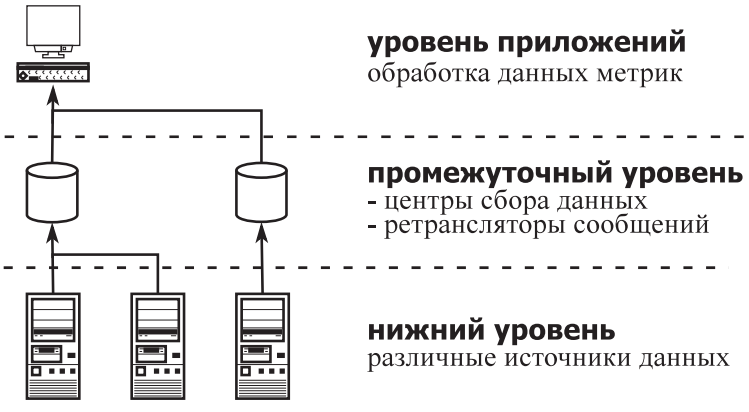

Рис. 1. Три уровня системы мониторинга

Благодаря возможности совместной работы приложений на разных платформах в тестировании разработанных модулей системы участвовали ОС MS Windows XP и различные версии дистрибутивов Linux (RedHat, CentOS).

Программный комплекс grate состоит из нескольких независимо функционирующих модулей, написанных на языке java: grated (аналог сервиса gmond системы Ganglia, способный замещать его на промежуточном уровне), grape (java-апплет, выполняющийся в среде web-браузера), grate-client (приложение для простой визуализации и контроля данных). Модули используют общую библиотеку интерфейсов и классов, позволяющую им работать с метриками, триггерами и прочими объектами.

Разработанная система мониторинга является трехуровневой [13], т.е. весь программно-аппаратный

комплекс строго разделен на три уровня (рис. 1). Передача данных может осуществляться лишь между соседними уровнями, позволяя изменять, дополнять и расширять уровни без нарушения 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 или данные сервиса gmond. Функциональность этого уровня мала, в целях более эффективного использования ресурсов контролируемого узла используется машинно-зависимое программное обеспечение (ПО).

Промежуточный уровень отвечает за сбор данных с нескольких источников, преобразование их к унифицированному формату данных, проверку простейших триггеров и выполнение соответствующих им действий. На этом же уровне функционируют ретрансляторы событий. Этот уровень можно реализовать на машинно-зависимых или интерпретируемых языках.

Уровень приложений отвечает за анализ данных и представление их пользователю. Быстродействие на таком уровне не так важно, поскольку выполняющееся на нем программное обеспечение может работать за пределами вычислительного комплекса, обращаясь лишь к данным, предоставляемым промежуточным уровнем.

Помимо функциональных возможностей, связанных с мониторингом, система grate предоставляет на всех уровнях возможность интеграции с другими программными комплексами через строго определенные интерфейсы взаимодействия либо напрямую с использованием библиотеки классов grate, либо через систему уведомлений и триггеров (см. ниже).

3. Компоненты системы мониторинга. Комплекс grate для получения данных использует сенсоры (абстрактные источники данных) уже установленных систем мониторинга, чтобы сократить объем необходимого программного обеспечения, устанавливаемого на вычислительный узел.

Данный подход позволил максимально использовать возможности развернутой ранее сторонней системы мониторинга (Ganglia) и реализовать дополнительные возможности, недоступные в этой системе без кардинальных изменений уже существующего программного обеспечения. При этом возможно собирать данные с различных систем мониторинга и обрабатывать их централизованно.

Базовой единицей хранения данных является метрика. Метрика — это набор значений определенного типа, отражающий изменения характеристики на конкретном узле. Метрики бывают статические, т.е. неизменяемые во время работы системы (версия операционной системы на вычислительном узле, объем физической памяти), и динамические (время работы узла, объем доступной физической памяти). Кроме того, метрики делятся на метрики с накоплением, которые обеспечивают доступ ко всем данным измерений, и одномоментные — хранят только текущее значение. Программный комплекс отслеживает непротиворечивость данных по времени, соблюдение порядка помещения данных в метрику. Метрика в общем случае хранит лишь набор данных за относительно небольшой промежуток времени и организует эффективный доступ к ним. К остальным данным, хранящимся во внешнем хранилище данных, также возможен доступ.

Все значения, получаемые во время мониторинга, периодически сохраняются в выбранном формате данных: по умолчанию — двоичный файл, база данных jRobin (http://www.jrobin.org/), являющаяся java-аналогом RRDTools, или любой пользовательский формат, если пользователь реализует соответствующий интерфейс и укажет его в конфигурационном файле при запуске центра сбора данных. Система также предоставляет возможность не сохранять значения метрик, теряя тем самым данные, поступившие ранее. Это может быть полезно, если анализируется метрика, значение которой важно лишь в момент ее измерения.

Метрика существует в рамках узла. В обсуждаемой архитектуре следует отличать вычислительный узел (от англ. host) от узла иерархической структуры данных (от англ. node). Далее в тексте под узлом понимается второе значение. Узел имеет изменяемые во времени атрибуты и набор метрик. Отличие атрибута от метрики в том, что атрибут всегда хранит только одно значение и не имеет явно указанного типа. Атрибут — это статичная одномоментная метрика строкового типа. Атрибутами являются вспомогательные данные, используемые при работе с системой, например, ими могут являться строки параметров командной строки, переданные приложению пользователя.

Узел всегда имеет родительский узел (кроме корня иерархической структуры), а также может иметь набор дочерних узлов. Grate динамически формирует иерархическую структуру, узлом которой может выступать любая определяемая источником данных сущность. Применительно к мониторингу кластеров могут быть сформированы метрики GRID (например, число кластеров), метрики кластеров (число узлов) и др. При этом метрике сопоставляется символьное имя, схожее с используемым в технологии XPath для доступа к узлам XML-документов. Это имя используется в определениях триггеров и для упроще-

ния доступа сторонними приложениями к метрикам. Подобный подход к иерархической организации используется в Ganglia и возможен в Nagios. Другие общедоступные системы мониторинга вычислительных кластеров в целом менее функциональны и не структурированы в иерархии. Система Supermon [14] схожа с Ganglia по архитектуре, однако не позволяет динамически добавлять узлы в контролируемую систему и не хранит историю изменения данных.

При иерархической организации системы сбора данных каждый центр данных хранит метрики, принадлежащие некоторому подмножеству узлов. Сборка всех метрик в единое дерево в grate осуществляется корнем иерархии центров сбора данных, что позволяет снизить потребление ресурсов на промежуточном уровне. Однако обсуждаемая архитектура не исключает и механизма обмена данными метрик между центрами сбора данных. Следует отметить, что, обладая хорошей масштабируемостью при иерархической организации сбора данных, Ganglia и некоторые другие системы мониторинга не позволяют организовывать структуры передачи данных, отличные от иерархических.

Последовательность преобразования данных от источников данных в метрики приведена на рис. 2.

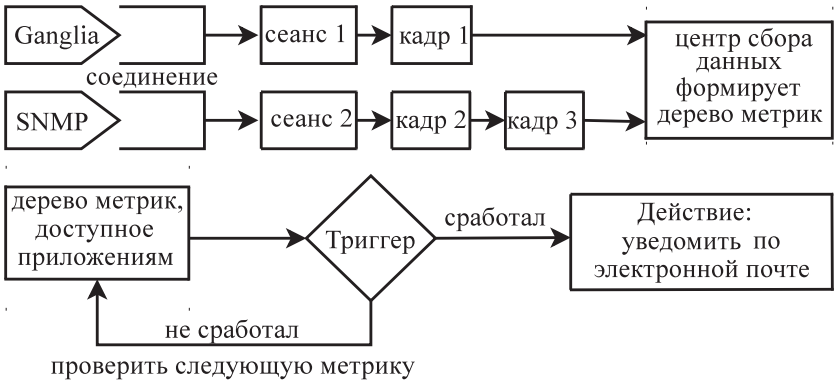

Рис. 2. Организация сбора данных

Получение данных осуществляется посредством соединений. Соединение — это инструмент, с помощью которого обеспечивается доступ к данным посредством сетевых соединений, открытия локального файла, доступа к базе данных и т.д. Результатом работы соединения является возможность открытия сеансов.

Соединение можно рассматривать как реализацию механизма получения потока необработанных данных от сенсора, к которому единожды при инициализации устанавливается подключение. В дальнейшем приложение инициирует открытие сеанса получения данных, используя ранее настроенное соединение. Работа с сеансом может привести к созданию последовательности нескольких кадров. Возможен также вариант, когда не будет создано ни одного кадра.

Каждый источник данных может измерять множества метрик, возможно, не одинаковые для различных источников. Через определенные промежутки времени источник данных делает мгновенный снимок состояния метрик и передает данные в процессе сеанса обмена данными на управляющий сервер, ответственный за обработку информации. Источник данных работает на нижнем уровне архитектуры, управляющий сервер — на промежуточном уровне или на уровне приложений.

Сеанс — это объект, идентифицирующий итерации получения данных. Обычно сеансы формируются центрами сбора данных.

Кадр — это набор данных, характеризующих метрики и значения в определенный момент времени. В общем случае в одном кадре содержатся данные для нескольких метрик. Следует отметить, что полученные значения полагаются неизменным до получения следующего кадра. Кадр есть логическое представление данных, в общем случае не зависящее от типа соединения.

Центр сбора данных может проводить агрегирование данных и создавать данные специальной метрики на основе набора однотипных метрик: например, метрика общей загрузки вычислительного кластера использует данные об использовании процессоров всех узлов вычислительной системы. Формирование метрики производится объектом-агрегатором метрик. При наличии соответствующих навыков пользователь системы может создать необходимые ему агрегаторы метрик в дополнение к уже имеющимся.

Следующими важными структурными элементами архитектуры обсуждаемой системы мониторинга являются триггер и действие. Триггер — это условие или выражение, истинность которого необходимо проверить для последнего значения метрики. Если условие выполнено, то состояние триггера изменяется, что приводит к выполнению заранее определенных действий.

Действие — набор инструкций, реакция на срабатывание триггера. Действие может широко варьироваться: запись диагностического сообщения в журнал, посылка уведомления администратору кластера, запуск приложений, проверка еще одного триггера и др. Одному триггеру может быть сопоставлено

несколько действий и, наоборот, одному действию может быть сопоставлено несколько триггеров. На рис. 3 приведена возможная схема проверки значений метрики, в которой условия Триггера 2 проверяются только после успешного выполнения условий Триггера 1, т.е. схема представляет собой реализацию логического элемента “И”. Пунктиром с Триггером 1 связано дополнительное действие, которое также инициируется при выполнении условий указанного триггера. Реализация элементов “ИЛИ” и “НЕ” затруднительна, поскольку действия, во-первых, выполняются только при успешном срабатывании триггера и, во-вторых, не предназначены для хранения значений метрик в целях использования одного и того же объекта-действия для нескольких различных метрик. В случае “НЕ” возможно написать триггер с условиями, обратными исходным. В случае “ИЛИ” логика будет реализовываться в действии, что ведет к усложнению соответствующего программного кода, нарушает архитектуру системы и в общем случае неэффективно. Для решения этой проблемы были введены агрегаторы триггеров, описание которых приведено ниже.

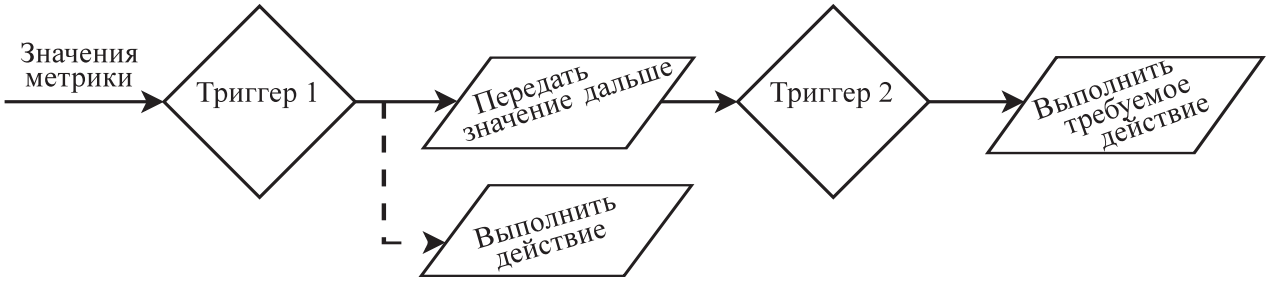

Рис. 3. Связка триггер-действие

Триггеры делятся на переключаемые и сбрасываемые. Первые находятся в сработавшем состоянии до тех пор, пока условие выполняется, и не производят действий повторно. Вторые выполняют действие каждый раз, если выражение, используемое в определении условия, истинно. Переключаемые триггеры имеет смысл использовать для генерации однократных уведомлений (например, при перегреве оборудования единственный раз послать электронную почту системным администраторам). Сбрасываемые триггеры удобны для выполнения преобразований над значениями метрики, удовлетворяющими условиям триггера (например, для формирования новой метрики, являющейся вычисленными значениями указанной пользователем функции). В таком случае каждое значение метрики должно быть проверено, независимо от того, удовлетворялись ли предыдущие условия или нет.

Основная задача триггеров — проверка выполнения простых условий для данных на предварительном этапе их анализа. Предполагается, что проверка триггеров проводится в основном на стороне сервера или на уровне центра сбора данных, поскольку проверка условий может быть относительно сложной вычислительной задачей. Все триггеры реализуют определенный в коде программного комплекса интерфейс, поэтому пользователь может реализовать свои собственные триггеры в случае необходимости каких-либо сложных алгоритмов проверки условий. Наиболее часто используемыми триггерами являются операции сравнения (больше/меньше), условие попадания/выхода из интервала, триггер проверки монотонности значений (возрастание/убывание). Последний триггер проверяет несколько предыдущих значений метрики, чтобы определить, выполнено ли условие монотонности.

Каждый триггер проверяет значения метрик независимо от других триггеров, а в случае сбрасываемых триггеров — независимо от предыдущих значений метрик. При этом возможно использование единственного объекта-триггера для проверки условий, наложенных на однотипные метрики. При большом числе метрик данный подход позволяет более эффективно использовать ресурсы оперативной памяти вычислительного узла, на котором выполняется проверка условий. Триггер для проверки условий может запрашивать значения метрик за любой промежуток времени, однако для большинства триггеров достаточно лишь получать последнее значение.

В ряде случаев необходимы логические операции над несколькими триггерами, например, выполнение действия, только если одновременно сработали три триггера. Возможна проверка всех трех через инициацию объектами-действиями последовательных проверок триггеров (такой подход приведен на рис. 3), однако более эффективным решением является введение объектов-агрегатаров, которые включают в себя триггеры или другие объекты-агрегаторы, каждый из которых проверяет выполнение логических операций “И”, “ИЛИ”, “НЕ” над набором указанных триггеров.

На рис. 4 изображены примеры агрегации двух триггеров. Слева — каждый триггер проверяет значение своей метрики, а действие выполняется агрегатором, который реализует логику “ИЛИ”. Справа —

оба триггера проверяют значения одной и той же метрики и агрегатор исполняет действия в случае срабатывания обоих триггеров. Система мониторинга гарантирует, что оба триггера будут проверять значения метрики для одного и того же временно́го интервала. Таким образом, результат работы агрегатора триггеров всегда корректен. Пользователь системы может реализовать свои агрегаторы для более сложной логики, если для его задачи недостаточно базовых логических элементов. Использование агрегаторов позволяет значительно эффективнее расходовать ресурсы, исключая при этом введение псевдодействий, единственной задачей которых является передача значения метрики следующему в последовательности триггеру.

Проверка значений метрик триггерами через равные промежутки времени потребляет сравнительно много ресурсов, не исключая избыточных проверок одного и того же значения несколько раз подряд, если новые данные еще не успели поступить. Более эффективным является подход, при котором условия триггера проверяются только при поступлении нового значения в метрику. Однако это порождает неопределенность в работе агрегаторов, если одна из метрик, проверяемая его триггерами, еще не содержит новых данных, а остальные уже получили их. В реализации нашей системы мониторинга агрегатору можно указать, как действовать в подобном случае: проверять значения, когда все метрики будут обладать необходимыми данными, или полагать, что у метрик с необновленными данными значения остались неизменными. По умолчанию выбирается второй вариант как быстрее реагирующий на изменения в контролируемых метриках.

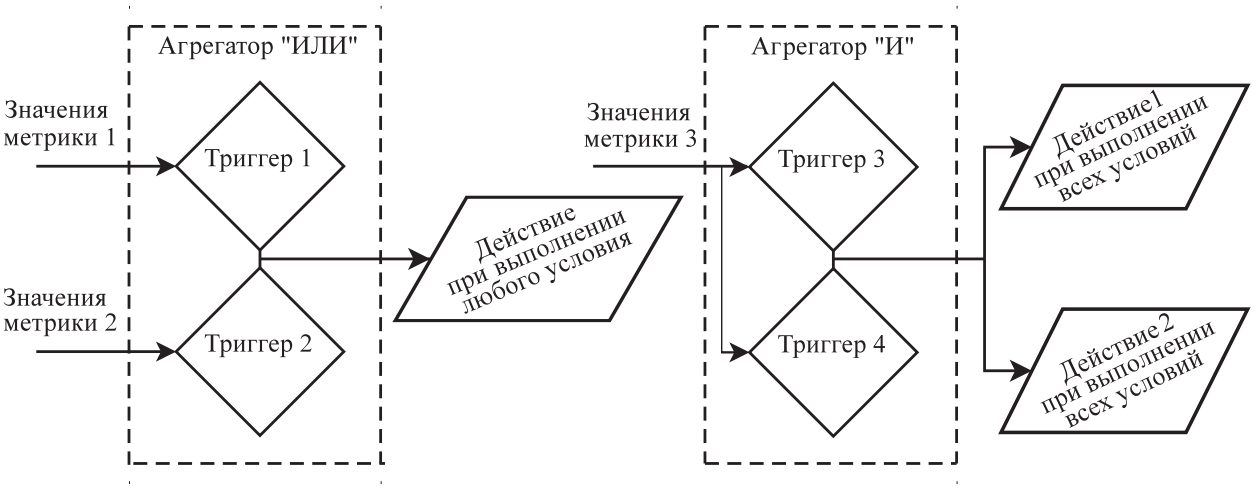

Рис. 4. Агрегаторы триггеров

Генератор триггеров (или метатриггер) — объект, создающий триггеры для метрик, подходящих под условия. Необходимость создания этого компонента продиктована динамической структурой иерархии узлов системы мониторинга, при этом часть узлов уже может быть создана (и иметь активные триггеры), а другая — отсутствовать и подключаться по необходимости. Метатриггеры облегчают добавление новых узлов в вычислительную систему тем, что позволяют создавать триггеры на базе указанного шаблона без вмешательства пользователя или администратора. В рассматриваемой системе мониторинга использование масок идентификаторов триггеров при описании агрегатора в конфигурационном файле позволяет динамически добавлять созданные триггеры в агрегатор при появлении новых метрик, еще более упрощая тем самым процесс настройки системы мониторинга.

Динамическая структура системы мониторинга порождает проблемы, одной из которых — в случае метатриггеров — является сложность удаления неиспользуемого триггера с целью освободить ресурсы системы. Основная проблема состоит в определении того, что триггер больше не будет использоваться (например, задача пользователя завершилась и условия триггера никогда не выполнятся). Существует несколько подходов к решению этой проблемы. Первый состоит в том, что пользователь явно уведомляет систему о том, когда созданные триггеры больше не являются необходимыми. Однако это не решает проблемы с не зависящими от пользователя триггерами и ставит систему в зависимость от пользователя, который может игнорировать необходимость освобождения ресурсов. Второй подход заключается в введении понятия тайм-аута метрики, т.е. такого временно́го интервала, за который обязательно должно прийти как минимум одно новое значение метрики. Если данные не поступают, то метрика считается неактивной и проверка ее значений не требуется; следовательно, связанные с нею триггеры можно удалить. Если же данные снова начнут поступать, то метатриггер создаст копию ранее удаленного триггера. В

обсуждаемой системе мониторинга был реализован последний подход.

Во многих широко распространенных системах, например в системе Ganglia, отсутствует возможность использования механизма триггеров и уведомлений о событиях. В ранних версиях разработанной системы мониторинга [15] для уведомления пользователей о событиях использовался только механизм триггеров, однако несмотря на гибкость данного подхода, у него имеются следующие недостатки: необходимость для пользователя реализовывать на уровне проверки триггеров нестандартный способ уведомления, невозможность структурированной и иерархической обработки события и другие, менее важные.

В настоящее время описываемая система мониторинга усовершенствована и относится к системам с дискретными событиями на базе подписки об уведомлении. Новая архитектура уведомлений о событиях позволяет устранить перечисленные недостатки. Каждый контролируемый узел системы мониторинга может генерировать уведомление о возникшем событии, в котором содержатся данные о событии. Это уведомление поставляется в рамках автоматически создаваемой метрики с типом данных XML-объект. Эта метрика может поставляться как по обычному соединению, по которому передаются кадры данных, так и по выделенному. Если центр сбора данных осуществляет сохранение истории таких метрик, то метрика события, как и всякая другая метрика, будет сохранена для дальнейшего анализа.

Пользовательские приложения подписываются (регистрируются) на ретрансляторе событий, которым обычно выступает центр сбора данных. При подписке указывается, в каких событиях заинтересовано приложение и в каком формате предоставлять информацию о событии. Основным способом является передача сообщения подписавшемуся приложению в том же формате, в каком оно было получено ретранслятором, т.е. в формате XML. Таким образом, при подписке создается фильтр уведомлений на ретрансляторе событий.

Центры сбора данных получают данные о событии и определяют необходимость уведомления подписчиков, если событие подходит под ранее установленный фильтр. При наличии зарегистрировавшегося приложения ему отправляется уведомление. Если центр сбора не имеет подписки на это уведомление, то сообщение при необходимости ретранслируется на другие центры сбора данных с целью проверки подписки там.

Клиентское приложение может обработать события, вернуть их неизменными для дальнейшей обработки на ретранслятор или же генерировать новые события. Например, при возникновении события отказа сетевой платы подписавшееся приложение может сгенерировать более конкретные события — “необходимость миграции задач с узла”, “необходимость уведомления администратора”, “необходимость удаления узла из списка ресурсов вычислительного кластера”, которые, в свою очередь, могут быть обработаны другими подписчиками.

Если подписок на одно и то же событие несколько, то все подписавшиеся приложения получат уведомление в порядке, зависящем от очередности времени подписки. Таким образом, если администратор кластера и технический специалист подписаны с помощью клиентского ПО на событие “необходимость уведомления администратора”, то оба получат соответствующие сообщения, например по электронной почте.

Другие системы мониторинга, такие как Nagios, позволяют производить обработку событий и поддерживают макроподстановки, которые можно использовать при обработке событий. События могут быть специфичными для узла и для сервиса, а также локальными и глобальными. Все обработчики событий указываются в конфигурационном файле до запуска системы, подписка на события во время работы системы мониторинга невозможна. Система Nagios изначально создавалась для мониторинга сетевых сервисов с жестко заданными событиями-состояниями для конкретных сервисов, поэтому добавление новых пользовательских событий затруднено.

Приложения, занимающиеся анализом метрик, выполняются на третьем уровне обсуждаемой архитектуры и принимают данные в структурированном виде от центров сбора данных посредством явного запроса отчета. В зависимости от реализации запрос может содержать указания по агрегированию необходимых данных: усреднить данные по всем узлам, показать суммарную информацию и др. В ряде случаев подобные данные автоматически формируют специальную метрику при получении новых значений центром сбора, если определен агрегатор метрик. Тогда приложение может запрашивать эту конкретную метрику, в других же случаях механизм отчета должен принять запрос в определенном формате и предоставить отчет на основе имеющихся данных.

Запрос принимается сервером отчетов по протоколу TCP, который передает этот запрос объекту-отчету. После формирования отчета сервер выдает полученные данные клиенту. На этом этапе также возможно включение пользовательских модулей формирования отчетов, что позволяет разработчикам стороннего ПО получать данные в наиболее удобном для них формате.

Разработанный программный комплекс grate-client запрашивает данные в формате XML и предоставляет пользователю информацию в графическом и табличном виде.

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

На кластере ВЦ ДВО РАН [16] разработанная в первом варианте система мониторинга использовалась также для контроля негативных изменений повышения температуры вычислительных узлов, например, в случае иногда возникающего отказа активного охлаждения. Серверная часть реализовала бóльшую часть функциональности gmetad и была настроена на сбор данных с использованием сервисов gmond, предоставляемых системой Ganglia с предварительно настроенной пользовательской метрикой температуры. При этом проводится анализ метрик, проверка триггеров, доступны также другие возможности системы. При превышении заранее указанной величины температуры происходила запись в журнал событий и отправка уведомления администратору кластера. В критическом случае принудительное отключение узла инициировалось аппаратными средствами, однако сигнал о необходимости отключения вычислительного узла также можно использовать для программного отключения средствами операционной системы.

Выполненное при мониторинге восьми узлов тестирование экспериментального кластера, а затем при моделировании контроля 100 узлов по методике, используемой разработчиками Ganglia, показало, что производительность комплекса (модуля grated) чуть выше чем у системы Ganglia 2.4 (модуля gmetad), уступая ей в объемах используемой памяти на управляющем сервере [17]. Модуль grated проводил проверку триггеров (по одному на каждый узел), расширяя тем самым базовую функциональность используемых сенсоров gmond. Визуализация выполнялась grate-client на основе данных, поставляемых от grated.

Тестирование работы grated и gmetad проводилось на управляющем узле кластера ВЦ ДВО РАН под управлением WhiteBox Linux 3.0 с ядром 2.4.20-8. Результаты приведены в табл. 1, где показаны значения общего времени выполнения T_0 и относительное время загрузки процессора T_p на управляющем узле. Использовалась система Sun JRE 1.5.

Таблица 1

Использование процессорного времени

Приложение	Кластер, 8 узлов		Эмуляция кластера, 100 узлов	
	T_0 (час.)	T_p (%)	T_0 (час.)	T_p (%)
grated	768	0.14	50.8	9.19
gmetad	888	0.23	192	10.6

При масштабировании системы неизбежно возникнет проблема роста суммарных по всем узлам затрат вычислительного времени, необходимого для обработки данных мониторинга. Для снижения возникающих издержек разработанная система позволяет использовать иерархические структуры сбора и обработки данных. Процесс мониторинга можно разделить на две логически несвязанные стадии:

1) получение, сохранение данных и обеспечение доступа к ним для нужд сервисов системы мониторинга;

2) обработка и проверка триггеров.

Эти стадии могут выполняться на различных уровнях иерархии и различных узлах вычислительного комплекса, в том числе на выделенных специально для этих целей.

Система позволяет организовывать потоки данных в структуру, в которой существует несколько центров сбора и обработки данных, требующих значительных вычислительных ресурсов для перевода значений метрик в унифицированный формат для дальнейшего анализа. После предварительной обработки объем данных может стать меньше за счет отсеивания ненужных, повторных либо неизменных значений. Затем значения метрик передаются выше по иерархии, где подвергаются дополнительной обработке. Таким образом, каждый уровень иерархии выполняет лишь часть задач по работе с данными, которая распределена между несколькими узлами. Это снижает нагрузку на каждый узел и повышает возможности масштабирования.

Более эффективными в задачах балансирования вычислительной нагрузки являются системы с распределенной обработкой данных без явных центров. Рассматриваемая архитектура не может использоваться в качестве полностью распределенной, поскольку связи между центрами сбора данных относительно жесткие и не могут свободно изменяться в процессе работы системы. Однако иерархическая

структура обладает значительными возможностями по масштабированию. В частности, показано, что аналогичные системы мониторинга способны поддерживать до 2000 узлов при организации передачи данных способом, схожим с описанным выше [2]. При этом рост суммарного использования процессорного времени на обработку данных ожидается несколько выше линейного, поскольку в связи с ростом числа передаваемых данных неизбежно будут возникать потери и задержки пакетов данных, что в итоге приведет к увеличению использования процессорного времени на обработку этих ситуаций.

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

В ряде случаев приложения пользователей кластера занимают не все вычислительные ресурсы узла, на котором выполняются, в то время как более емкие 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 [18] была добавлена возможность миграции 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 — гостевыми. Миграция выполняющейся копии операционной системы доступна с использованием стандартных средств XEN. Дополнительная возможность создания слепка выполняющейся операционной системы (так называемый checkpoint) может использоваться для восстановления работоспособности при сбое в работе вычислительного кластера или отключении электроэнергии.

Для тестирования XEN и его взаимодействия с grate был создан отдельный логический кластер в составе экспериментального кластера, по пакету программ идентичный основному кластеру, но с модифицированным для поддержки XEN ядром Linux и установленными утилитами управления системой виртуализации. Во время тестирования осуществлялся процесс миграции выполняющейся копии операционной системы с запущенными вычислительными задачами с одного из физических узлов на другой, обладающий бóльшими доступными ресурсами на момент миграции.

Программный комплекс анализировал эффективность использования ресурсов на вычислительном узле на основе данных, предоставляемых промежуточным уровнем системы grate (эффективная загрузка процессора, объем занятой памяти). При необходимости триггером инициировалось выполнение shell-скрипта с командами миграции, при этом не происходило значительных прерываний вычислительных процессов. Как показали тестовые замеры, для малого числа узлов, участвующих в расчете параллельной задачи с интенсивной передачей данных между узлами, потери были малы. В табл. 2 приведены результаты для трех узлов. В испытаниях использовались задачи теста HPL (High-Performance Linpack: параллельный вариант теста Linpack), что позволило одновременно провести измерение производительности вычислений, результат усреднялся на основе серии из 10 испытаний. Следует отметить, что тест HPL достаточно интенсивно использует коммуникационную сеть. Для задач, в меньшей мере зависящих от интенсивности обмена данными между вычислительными узлами, потери производительности будут меньше. При отсутствии миграций использование XEN не приводит к заметным потерям производительности.

Таблица 2

Потери производительности при работе XEN

Число миграций в час	12	6	3	2	0
Производительность в GFlops	8.8	9.7	10.6	10.7	11.0
Относительные потери производительности в %	19.7	11.5	3.9	2.9	—

Важным является вопрос надежности и отказоустойчивости разработанной системы мониторинга. Программный комплекс grate позволяет организовывать иерархическую структуру центров сбора данных с дублированием центров. Поскольку данные от сенсоров в основном передаются широковещательно, то это не приводит к дополнительным накладным расходам для вычислительных узлов. Поддержка непротиворечивости данных каждым центром, в том числе корнем иерархии, позволяет избежать дублирования значений метрик при одновременной работе центров.

Описанная архитектура при соответствующей настройке системы grate позволяет создавать такую организацию сети, при которой каждый центр сбора данных обладает всей информацией о вычисли-

тельной системе. Выход из строя одного из центров не приводит к потере информации и нарушению работоспособности системы. Однако при таком подходе растет число обменов данными, и в результате общая нагрузка на сетевую подсистему увеличивается, что затрудняет масштабирование системы мониторинга.

Модули уведомления о событиях и генерации отчетов поддерживают авторизацию с использованием пары "имя пользователя – пароль", передающейся в незакодированном виде. Разработанная система в настоящее время не предоставляет общего механизма для авторизации доступа ко всем своим сервисам. Следует также отметить, что выбранный способ авторизации является ненадежным и редко используется в небезопасных средах передачи данных. Однако в настоящей работе предполагается, что основные функции обеспечения безопасности переданы операционной системе и системному программному обеспечению, исключающим проникновение злоумышленника в сеть системы (ограничение доступа по IP-адресу, системы обнаружения сканирования TCP/IP портов и др.).

Использование языка программирования java, поддерживающего автоматическую сборку "мусора", исключает утечки памяти. Выполнение программы в рамках виртуальной машины позволяет не нарушать работу вычислительного комплекса даже при возможном появлении в системе мониторинга неустранимой ошибки. За все время тестирования на кластере ВЦ ДВО РАН в работе системы grate не было никаких ошибок, повлекших за собой нарушение работы вычислительной системы или потерю данных.

5. Заключение. Разработанное по трехуровневой архитектуре приложение показало сравнимые результаты оценки производительности с уже имеющимися решениями, расширив функциональность используемой на нижнем уровне системы мониторинга. Система может быть применена для работы в составе различных программных комплексов для решения разнообразных задач.

Использование языка java позволило сделать систему гибкой и легко расширяемой: для добавления нового триггера или действия достаточно реализовать интерфейс, определенный в исходном коде системы мониторинга. Созданный таким образом модуль можно использовать, не останавливая работу системы.

Программный комплекс grate предоставляет унифицированный подход к обработке событий, позволяя приложению подписаться на необходимые события уже во время работы системы. Возможна также обработка событий с помощью действий и триггеров с макроподстановками и указанием необходимых действий в конфигурационном файле. Таким образом, система мониторинга grate в большей степени расширяема, чем аналоги: на всех уровнях динамически формирует внутренние структуры и содержит более широкий набор базовых сервисов (система уведомлений, триггеры, механизм отчетов).

Описанная нами система использует жестко заданный путь прохождения события, не обязательно зависящий от его данных. Следует отметить, что механизм событий, предлагаемый в настоящей статье, более масштабируем, чем используемые подходы в системах XMLBlaster и ISIS.

В созданной реализации системы мониторинга данные предоставляются пользователю через механизм формирования отчетов, получаемых после запроса к конкретному центру сбора данных, при этом формат запроса может быть произвольно изменен пользователем системы. В одном отчете могут содержаться метрики из различных систем мониторинга, развернутых на нижних уровнях, а также данные аппаратных сенсоров.

Построение программных комплексов, реализующих предложенную архитектуру мониторинга, упрощает создание других систем контроля и управления вычислительными системами.

СПИСОК ЛИТЕРАТУРЫ

1. *Barth W.* Nagios system and network monitoring. San Francisco: No Starch Press, 2006.
2. *Massie M.L., Chun B.N., Culler D.E.* The Ganglia distributed monitoring system: design, implementation, and experience. Berkeley: Berkeley Univ. Press, 2003.
3. *Mukherjee B., Heberlein T.L., Levitt K.N.* Network intrusion detection // IEEE Network. 1994. N 8. 26–41.
4. *Spafford E.H., Zamboni D.* Data collection mechanisms for intrusion detection systems. CERIAS Tech. Rep. West Lafayette: Purdue Univ. Press, 2000.
5. *Писарев А.В., Пересветов В.В.* Нейросетевые компоненты мониторинга вычислительного кластера // Тр. конференции "Информационные и коммуникационные технологии в образовании и научной деятельности". Хабаровск: Изд-во Тихоокеанского гос. ун-та, 2008. 319–323.
6. *Benveniste A., Fabre E., Jard C., Haar S.* Diagnosis of asynchronous discrete event systems, a net unfolding approach. IRISA Tech. Rep. RR-4181. Rennes, 2001.
7. *Yingquan W., Christoforos N.H.* Distributed non-concurrent fault identification in discrete event systems // Proc. of Multiconference on Computational Engineering in Systems Applications. Lille, 2003.
8. *Birman K.P., Joseph T.A.* Exploiting virtual synchrony in distributed systems // Proc. of the 11th ACM Symposium on Operating Systems Principles. Austin, 1987. 123–138.

9. *Aguilera M., Strom R., Sturman D., Astley M., Chandra T.* Matching events in a content-based subscription system // Proc. of the 18th ACM Symposium on Principles of Distributed Computing. Atlanta, 1999. 53–61.
10. *Snoeren A., Conley K., Gifforfd D.* Mesh-based content routing using XML // Proc. of the 18th ACM Symposium on Operating Systems Principles. Banff, 2001. 160–173.
11. *Bonnet P., Gehrke J., Seshadri P.* Towards sensor database systems // Proc. of the 2nd International Conference on Mobile Data Management. Hong Kong, 2001.
12. http://sourceforge.net/projects/grate
13. *Тарасов А.Г.* Трехуровневая система мониторинга расширенной функциональности // Тр. международной конференции "Параллельные вычислительные технологии". Челябинск: Изд-во ЮУрГУ, 2008. 464–469.
14. *Scottie M., Minnich R.* Supermon: a high-speed cluster monitoring system // Proc. of IEEE Cluster Computing. Chicago, 2002. 39–46.
15. *Тарасов А.Г.* Мониторинг вычислительного кластера с использованием java-технологий // XXX Дальневосточная математическая школа-семинар имени акад. Е. В. Золотова. Хабаровск: Изд-во ДВГУПС, ИПМ ДВО РАН, 2005. 201.
16. *Пересветов В.В., Сапронов А.Ю., Тарасов А.Г.* Вычислительный кластер бездисковых рабочих станций. Препринт № 83 ВЦ ДВО РАН. Хабаровск, 2005.
17. *Пересветов В.В., Сапронов А.Ю., Тарасов А.Г., Шаповалов Т.С.* Удаленный доступ к вычислительному кластеру ВЦ ДВО РАН // Вычислительные технологии. Т. 11. Новосибирск: Изд-во ИВТ СО РАН, 2006. 45–51.
18. *Пересветов В.В., Сапронов А.Ю., Тарасов А.Г., Шаповалов Т.С.* Организация работы вычислительного кластера в режиме удаленного доступа. Препринт № 110 ВЦ ДВО РАН. Хабаровск, 2007.

Поступила в редакцию
28.09.2008