УДК 004.457

К.С. Стефанов

Московский государственный университет
им. М.В. Ломоносова, Москва, Россия

СИСТЕМА МОНИТОРИНГА ПРОИЗВОДИТЕЛЬНОСТИ СУПЕРКОМПЬЮТЕРОВ

Системы мониторинга суперкомпьютерных комплексов – важная часть их программного обеспечения, призванная обеспечить максимально эффективное и бесперебойное функционирование, а в случае возникновения нештатных ситуаций – сохранность оборудования и оповещение персонала. Существующие системы мониторинга справляются с задачами обеспечения сохранности оборудования, хотя достижение нужного времени реакции для больших комплексов может быть непростой задачей. Кроме того, возникла еще одна важная область использования систем мониторинга – мониторинг производительности. Существующие системы предназначены для исследования отдельных конкретных задач или суперкомпьютера в целом и не могут быть использованы для исследования производительности всего потока задач, работающих на суперкомпьютере, и оценки эффективности использования суперкомпьютера в целом одновременно. Мониторинг производительности вычислительных комплексов в целом и отдельных программ – важная задача, решение которой позволяет оценить эффективность использования имеющихся ресурсов и предложить пути увеличения эффективности выполняемых программ. Рассмотрен подход к созданию системы мониторинга производительности, который позволит исследовать производительность всего потока задач, выполняющихся на вычислительном комплексе. Предлагаемый подход основан на возможности направления разных потоков данных по различным путям передачи, динамической реконфигурации режимов работы системы, обеспечении вычисления метрик производительности без промежуточного сохранения данных мониторинга на диск и переносе части обработки данных на вычислительные узлы суперкомпьютера. Рассмотрены детали реализации системы мониторинга, основанной на указанном подходе, и приведены результаты измерения производительности разрабатываемой системы.

Ключевые слова: вычислительные системы, суперкомпьютер, мониторинг, мониторинг суперкомпьютеров, мониторинг производительности, эффективность, поток задач, исследование потока задач, производительность, вычисление метрик производительности.

K.S. Stefanov

M.V. Lomonosov Moscow State University, Moscow, Russian Federation

A PERFORMANCE MONITORING SYSTEM FOR SUPERCOMPUTERS

Monitoring systems for supercomputers are the crucial part of supercomputer software. Its aim is to maintain effective and uninterruptible operation. In case of emergencies they must provide hardware safety and staff alerting. Existing monitoring systems suit for providing hardware safety, although achieving required response time may be difficult. Another use of monitoring system is performance monitoring. Existing monitoring systems are designed for analyzing of separate job or the complete supercomputer, and can't be used for analyzing of total stream for jobs being executed on supercom-

puter, and simultaneously compute performance metrics for computing system as a whole. In this paper we propose an approach to constructing performance monitoring system capable of analyzing every job executed on supercomputer. The proposed approach is based on capability of directing separate monitoring data flow along separate paths, dynamic reconfiguration of monitoring system, computing performance metrics without intermediate saving data to disk, and transferring a part of data processing onto compute nodes of a supercomputer. The details of implementation of a monitoring system based on proposed approach are provided, and the result of its performance testing are given.

Keywords: computing systems, supercomputer, monitoring, supercomputer monitoring, performance monitoring, efficiency, job stream, job stream analysis, performance, performance metrics computing.

Цели мониторинга суперкомпьютеров

Рассмотрим возможные причины и цели мониторинга суперкомпьютеров и опишем особенности систем, которые могут применяться в каждой области.

В таблице приведена сводка по возможным целям мониторинга и требуемым характеристикам.

Цели мониторинга суперкомпьютеров

Цель мониторинга	Требуемая надежность	Сложность обработки	Объем данных
Аварийные ситуации	Очень высокая	Низкая	Небольшой
Работоспособность узлов	Средняя	Средняя	Средний
Функциональность узлов и вычислительной инфраструктуры	Средняя	Высокая	Средний
Мониторинг эффективности	Низкая	Низкая	Высокий

Самая важная причина мониторинга – отслеживание аварийных ситуаций. Сюда относятся случаи срабатывания пожарной сигнализации и запуск автоматических систем пожаротушения, протечки в системе охлаждения, проблемы с электропитанием и другие происшествия, которые требуют немедленной реакции для сохранения оборудования и минимизации ущерба. Для обеспечения быстрой реакции системы мониторинга нужно везде, где возможно, использовать оповещения от датчиков наступления таких ситуаций, которые будут обрабатываться системой мониторинга в момент их прихода (пассивный мониторинг). В случае если используемые датчики не позволяют выдавать оповещения при наступлении событий, необходимо организовывать их опрос (активный мониторинг) с достаточно малым периодом (единицы секунд). При этом необходимо обеспечить высокую надеж-

ность обнаружения аварийных ситуаций, так как задержка или отсутствие правильной реакции могут привести к выходу из строя большого количества дорогостоящего оборудования. Поток данных, который нужно обрабатывать, небольшой, поскольку количество инфраструктурного оборудования и датчиков в нем невелико даже в больших комплексах.

Следующей причиной является отслеживание исправности (работоспособности) вычислительных узлов и их компонентов: вентиляторов, блоков питания и т.п. Своевременное отключение узла с неисправным вентилятором повышает шансы на его относительно быстрый ремонт. Необходимо иметь время реакции порядка единиц минут. Надежность здесь также важна, однако если в случае с аварийными ситуациями несвоевременная реакция может привести к потере значительного количества оборудования, то потери от несвоевременного обнаружения неисправностей на узлах обычно сводятся к самим этим узлам. Также стоит отметить, что современное оборудование обычно имеет встроенную защиту от некоторых видов таких неисправностей (перегрев, остановка вентиляторов). Поток данных здесь составляет единицы датчиков на каждый вычислительный узел.

Еще одна причина – отслеживание пригодности вычислительных узлов для исполнения счетных задач. Здесь контролируется отсутствие ошибок ECC, сбоев жестких дисков, ошибок на сетевых интерфейсах и т.п. Требуемое время реакции на появление таких ошибок составляет от единиц минут до часов, и проверка может проводиться не периодически, а, например, перед каждым запуском вычислительной задачи. Требования к надежности также невысоки, так как пропущенная ситуация такого рода не приводит к физической порче оборудования, а влечет только сбой или увеличенное время выполнения задач. При этом объем данных составляет несколько десятков датчиков на вычислительный узел.

Самые низкие требования по надежности предъявляются к мониторингу производительности. Мониторинг производительности – это отслеживание различных параметров ОС и оборудования для того, чтобы найти, что мешает выполняемой программе достигнуть более высокой производительности. Отказ этой подсистемы не приводит к критическим последствиям – задачи продолжают исполняться, оборудование не страдает, но возникают неудобства при отладке программ.

При этом для того, чтобы получаемая информация была полезной, период съема информации должен составлять единицы, а лучше доли секунд. Объем снимаемой информации составляет десятки датчиков на вычислительный узел и имеет тенденцию к росту, так как информация снимается в том числе с отдельных процессорных ядер, а их число в узле увеличивается.

При этом мониторинг производительности имеет одно существенное отличие от остальных видов мониторинга: объект мониторинга (вычислительная задача) не имеет постоянного выделенного «места» внутри суперкомпьютера, набор датчиков, относящихся к конкретной задаче, формируется перед ее запуском, в момент выделения под нее вычислительных ресурсов (узлов), что затрудняет выделение данных, относящихся к конкретной задаче, из общего потока.

Архитектура систем мониторинга

На рис. 1 изображена общая архитектура систем мониторинга. За получение информации отвечают агенты, запущенные на вычислительных узлах. Некоторые системы характеризуются как безагентные (agentless), например [1]. В этом случае роль агентов играют какие-то сервисы операционной системы, работающей на узле. Это могут быть сервера SNMP, доступ по протоколу SSH для получения информации и т.п. При получении информации о состоянии других компонент вычислительной системы вместо агентов на вычислительном узле используются агенты SNMP или компоненты, отвечающие за получение данных по каким-то другим протоколам. Собранные таким способом данные передаются в серверную часть системы мониторинга, где происходит их обработка, сохранение какой-то части информации в базу данных, выделение событий, реагирование на события и имеются части, предназначенные для представления данных пользователю, в частности визуализации.

Значительное количество существующих систем мониторинга [1–7] построено по описанной схеме. Остановимся на их общих особенностях.

Во-первых, это отсутствие какой бы то ни было обработки данных на вычислительном узле, являющемся источником этих данных. Причины этого очевидны для безагентных систем. Но и в системах, у которых имеются на вычислительных узлах собственные агенты, эти агенты только собирают необходимую информацию и передают ее дальше без обработки. Обычно такой подход объясняют необходимо-

стью уменьшить влияние работы агента на ход выполнения вычислительной задачи, которая работает на этом вычислительном узле.

Другой особенностью существующих систем мониторинга является жесткая конфигурация путей передачи данных. Пути передачи данных задаются при конфигурировании системы и не меняются в процессе работы. Для каждого агента задается единственный адрес в серверной части системы мониторинга (или несколько адресов, если серверная часть строится с дублированием), куда пересылаются собираемые данные, и этот адрес не меняется в процессе работы.

Для обеспечения необходимой производительности серверную часть системы мониторинга делают распределенной с древовидной структурой. Каждый листовой компонент серверной части отвечает за какую-то часть вычислительных узлов. В этом компоненте информация обрабатывается, агрегируется и выделяются события. Затем обработанный и уменьшенный поток информации передается на следующие уровни для дальнейшей обработки.

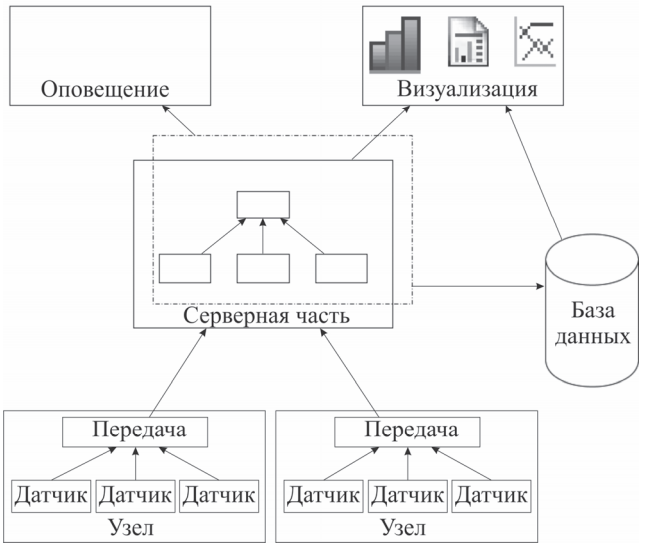

Рис. 1. Общая архитектура систем мониторинга

Для обеспечения надежности какие-то компоненты серверной части или она вся могут дублироваться на независимых серверах. При этом потоки данных от агентов на вычислительных узлах передаются сразу в две точки, и в дальнейшем весь путь также дублируется. При выходе из строя какого-то из компонентов его функции берет на себя дублирующий компонент.

Такая архитектура хорошо работает для задач отслеживания аварийных ситуаций (перегрев, протечки, проблемы с электропитанием, пожар), доступности отдельных компонентов вычислительной системы и их пригодности к работе. Период сбора данных при этом составляет от десятков секунд до десятков минут, а также используется пассивный мониторинг (реагирование на сигналы о событиях, приходящие от внешних источников).

Задача мониторинга производительности отличается от других задач мониторинга. Сбор данных должен производиться чаще: период должен составлять единицы секунд, а потенциально – доли секунд. Как следствие, объем данных, который нужно обрабатывать, значительно больше. Кроме того, вычислительная задача – объект динамический, ее расположение и источники данных, относящиеся к конкретной задаче, определяются в момент запуска этой задачи.

Предложено много походов к мониторингу производительности задач [8–16]. Они объединены тем, что предназначены для мониторинга производительности конкретных задач. С этой целью в момент запуска задачи производится запуск агентов на вычислительных узлах, где будет выполняться задача, данные от которых собираются в базу данных, а после окончания работы задачи производится анализ.

Этот подход вполне оправдан при анализе отдельных задач, однако если ставить целью анализ всего потока задач, исполняемых на вычислительной системе, то такой подход приносит множество лишних накладных расходов.

При анализе всего потока задач детальное рассмотрение каждой задачи невозможно, да и не представляет интереса. От каждой задачи будет использоваться небольшое число метрик производительности, которые будут подсчитаны один раз. Подробный анализ нужен только для отдельных задач, поэтому данные мониторинга от большинства задач будут один раз записаны в базу данных и один раз из нее прочитаны при подсчете соответствующих метрик.

Однако такой режим использования приводит к сильному падению производительности системы хранения данных, используемой базой данных, либо к ее удорожанию. Ведь для самых распространенных систем хранения данных на основе жестких дисков наиболее производительным режимом является режим линейной записи или линейного чтения. Однако одновременная запись данных мониторинга и чтение

данных для подсчета метрик завершившихся задач приводит к переходу в режим случайного доступа к данным, что для систем на основе жестких дисков дает падение производительности на два порядка и более. Решением могут стать системы на основе твердотельных накопителей (SSD), однако они гораздо дороже систем на жестких дисках и, кроме того, деградируют при большом количестве перезаписей, что обязательно будет иметь место при использовании их для хранения базы данных с информацией по производительности.

Кроме того, если задачей является анализ производительности вычислительной системы в целом, то запуск отдельной инфраструктуры для получения данных конкретной задачи приводит к тому, что фактически создаются две параллельные системы сбора данных: одна для оценки производительности вычислительной системы в целом, работающая постоянно, и вторая, запускаемая для конкретных задач.

Предлагаемый подход

Для решения вышеуказанных проблем в рамках единой системы мониторинга мы предлагаем систему, основанную на следующих принципах:

а) возможность направлять разные потоки данных по разным путям или копии одних и тех же данных нескольким получателям;

б) обеспечение динамической переконфигурации режимов работы системы мониторинга (путей передачи данных, параметров сбора данных, путей обработки данных);

в) обеспечение возможности вычисления метрик производительности для отдельных задач во время сбора данных, без записи их на диск с последующим считыванием;

г) перенос части обработки данных мониторинга на вычислительные узлы.

Направление разных данных по разным путям позволяет одновременно вычислять метрики как для системы в целом, так и для задачи, выполняющейся на конкретном вычислительном узле. Данные, на основе которых рассчитываются метрики вычислительной системы в целом, передаются на компоненты, ответственные за такой расчет, а данные, необходимые для вычисления метрик по отдельным задачам, передаются на компоненты, ответственные за эти задачи.

При этом для вычисления метрик по отдельным задачам необходимо уметь создавать компоненты для вычисления этих метрик в момент запуска задачи и выстраивать пути передачи данных таким образом, чтобы получать все необходимые для этого данные (и только их) в нужной точке, т.е. выполнять переконфигурацию компонентов системы мониторинга и путей передачи данных в процессе работы. Такая переконфигурация дает возможность производить вычисления метрик в процессе работы, без промежуточного сохранения в базе данных. Это требует также сопряжения системы мониторинга с менеджером ресурсов вычислительной системы.

Отметим, что для полноценного мониторинга производительности нужны данные не только от вычислительных узлов, на которых работает задача, но и общих элементов вычислительной системы: систем хранения данных, сетевых коммутаторов и т.п. [10, 14].

Наконец, вынос части обработки на вычислительные узлы позволит существенно увеличить производительность системы мониторинга и ее возможности по обработке информации. Обычно избегают какой-либо обработки информации на вычислительных узлах, мотивируя это необходимостью уменьшения влияния на выполняемую на узле задачу. Однако использование технологий типа Simultaneous multithreading (SMT), или Hyper-threading, как она называется в процессорах Intel, позволяет загрузить недоиспользованные основной задачей ресурсы процессора другой деятельностью с минимальным влиянием на основную задачу. Виртуальные ядра, доступные при включении SMT, как правило, не используются для выполнения вычислительных задач, так как обычно все нити вычислительной задачи используют одни и те же ресурсы процессора, поэтому запуск дополнительных нитей на виртуальных ядрах не приводит к увеличению общей реальной производительности. И даже если не использовать технологии типа SMT, а использовать небольшую часть ресурсов вычислительных улов так, чтобы оставить влияние на задачу небольшим, в сумме выйдет ресурс со значительной суммарной производительностью. Для получения сопоставимой производительности на выделенном оборудовании может понадобиться значительное количество серверов и сопутствующей инфраструктуры.

Архитектура разрабатываемой системы

Далее перейдем непосредственно к описанию разрабатываемой системы.

Система мониторинга разрабатывается для работы под управлением ОС семейства Linux, поэтому везде далее под операционной системой мы будем понимать именно ОС с ядром Linux.

Система мониторинга является распределенной и состоит из агентов мониторинга. Агент мониторинга с точки зрения операционной системы – это процесс, работающий на вычислительном узле или на другом компьютере (сервере). Агент мониторинга представляет собой систему исполнения – программное окружение, которое обеспечивает функционирование узлов, связей между ними и передачу сообщений.

Всё получение и обработка информации о состоянии вычислительной системы ведется в узлах системы мониторинга (далее под узлами мы будем понимать именно узлы системы мониторинга, а не вычислительные узлы суперкомпьютера). Данные передаются от узла к узлу. При этом каждый узел конфигурируется независимо от других. Связи между узлами создаются независимо друг от друга.

В системе передаются сообщения двух типов. Основной тип – сообщения, содержащие данные мониторинга. Они передаются от узла к узлу вдоль созданных связей между узлами. Второй тип – управляющие сообщения, они передаются непосредственно от одного узла к другому вне зависимости от наличия между ними связей.

Каждый узел имеет определенный тип. Тип узла задается при его создании. Тип определяет функции, которые вызываются системой исполнения, когда узлу надо выполнить какое-то действие. Количество узлов каждого типа не ограничено. Один или несколько типов описываются в модуле. Модуль представляет собой динамически подключаемую библиотеку с определенным интерфейсом. Какие именно модули подключаются при запуске агента мониторинга, задается его конфигурацией. В текущей версии реализации все модули загружаются исключительно при старте агента мониторинга. Набор типов узлов, из которых будет строиться функциональность агента мониторинга, определяется типами, описанными в загруженных модулях. Отметим, что ограничение на загрузку модулей исключительно при старте агента не является принципиальным и при необходимости в дальнейшем может быть снято.

Каждый узел имеет ноль или больше именованных входов, ноль или больше именованных выходов и может принимать и передавать управляющие сообщения. Все узлы имеют числовой идентификатор, присваиваемый системой исполнения при их создании. Кроме того, узлу может быть присвоено имя (строка символов).

При создании связи между узлами задаются узел – источник данных (при помощи имени или идентификатора), имя выхода этого узла для создания связи, узел – приемник данных и имя входа, на который данные будут передаваться. После создания связи копия данных, которые узел будет передавать на данный выход, будет передаваться системой исполнения на все входы, связанные с данным выходом.

Формат данных, передаваемых между узлами, представляет собой последовательность троек <Id, Len, Value>, где Id – целое беззнаковое число, определяющее тип данных; Len – длина данных (поля Value) в байтах; Value – сами данные. Интерпретация содержимого поля Value осуществляется узлами, обрабатывающими данные. В поле Value может содержаться одно значение, вектор значений (например, значения какого-то параметра для всех процессорных ядер на вычислительном узле) или какая-то другая структура данных. При этом узел может передавать данные без изменения даже в том случае, если конкретная семантика поля Value ему неизвестна. В одном сообщении с данными мониторинга может содержаться произвольное число таких троек.

Управляющие сообщения служат для создания и уничтожения узлов, задания их имен, создания и уничтожения связей между узлами, создания таймеров и подписки на них (см. ниже) – такие сообщения обрабатываются системой исполнения (агентом мониторинга). Сообщения, служащие для изменения настроек узлов, и другие, специфичные для конкретного узла (типа узлов), обрабатываются непосредственно узлами, которым они адресованы. Управляющее сообщение может быть послано любым узлом любому другому. Для определения получателя сообщения используется имя или номер узла-получателя.

Еще один элемент, необходимый для работы системы мониторинга, – таймеры. При создании таймера определяется его имя, а также моменты времени, в которые он будет срабатывать. Это может быть один предопределенный момент времени (абсолютный или относительно текущего времени), или таймер может срабатывать с заданной периодичностью. При срабатывании таймер посылает управляющее

сообщение всем подписанным на него узлам. При одновременном срабатывании нескольких таймеров посылается одно управляющее сообщение, содержащее список сработавших таймеров. Для подписки также используется управляющее сообщение. Узел может подписать на таймер самого себя или другой узел.

Всё взаимодействие внутри системы реализовано в функциональном стиле. Тип узла определяет функции, которые вызываются для обработки получения данных мониторинга, получения управляющих сообщений, при инициализации и уничтожении самого узла, создании связи и т.п. Эти функции могут быть переопределены для конкретного узла, а функции, связанные с обработкой данных мониторинга, – для конкретного входа.

Для построения системы мониторинга используется несколько видов узлов. Мы говорим о нескольких видах, но они отличаются лишь с точки зрения их функциональности, системой исполнения все узлы обрабатываются одинаково.

Узел, который получает данные мониторинга от программно-аппаратной среды (операционной системы или оборудования), называется датчиком. Датчик – это узел без входов. Его назначение – по сигналу от таймера получить данные мониторинга путем обращения к аппаратуре, сервисам ОС и т.п. и передать его на свой выход (у датчиков он обычно один).

Узел с входами и выходами обычно осуществляет обработку данных мониторинга. Обработка может быть любой: фильтрация данных, вычисление скорости изменения величины (как, например, вычисление скорости передачи и приема данных по счетчикам переданных и полученных байтов), сравнение с заданным порогом и т.д. Именование входов и выходов нужно для разделения потоков данных. Например, узел, отбирающий данные по какому-то критерию, может передавать данные, подходящие под критерий фильтрации, на выход с именем match, а данные, не подходящие под критерий, – на выход с именем notmatch. Соединив эти выходы с другими узлами, мы получаем разделение потоков данных, которые затем могут обрабатываться независимо. Или же имя входа может определять типы данных мониторинга, которые будут передаваться. Например, для узла, вычисляющего скорость изменения величины, в имени входа может быть закодирован тип этой величины (uint32, int64 и т.п.). При создании первой

связи с новым входом узел получает имя входа и может запретить создание связи, если не имеет информации о том, как обрабатывать информацию, получаемую через такой вход. Аналогичная проверка может производиться и при создании связи с выходами узла.

Узлы с входами, но без выходов предназначены для передачи данных мониторинга вовне агента. В данном случае мы имеем в виду отсутствие выходов с точки зрения системы исполнения, а не выход для передачи информации вообще. Чаще всего такой узел будет использоваться для организации взаимодействия агентов. Узел принимает данные мониторинга на своих входах, сериализует их и передает по сети другому агенту, работающему на другом компьютере или на том же самом. На входе принимающего агента будет работать узел-приемник, который во многом устроен как датчик (тоже не имеет входов), однако данные он получает не от аппаратуры или ОС, а принимает по сети от другого агента. Настройка приемников и передатчиков (куда передавать данные, на каком сетевом адресе принимать соединения, по какому протоколу производить обмен и т.п.) осуществляется путем посылки управляющих сообщений.

Другими вариантами узлов без выходов могут быть узлы, сохраняющие данные мониторинга во внешнюю базу данных или запускающие внешние реакции: отсылающие сообщение администратору по электронной почте, SMS, записывающие сообщение в журнал или же инициирующие выключение каких-то компонентов вычислительной системы.

Поскольку каждый датчик получает сообщение от таймера независимо от других и отсылает полученные им данные мониторинга в отдельном сообщении, то для эффективности необходимо собирать данные из нескольких таких сообщений в одно, прежде чем передавать их по сети. Такой сборкой занимается узел консолидации. Этот узел при получении любого сообщения с данными мониторинга сохраняет это сообщение у себя во внутреннем буфере и посылает системе исполнения специальный запрос (специальное управляющее сообщение). По этому запросу система исполнения создает список всех таймеров, которые активны в момент запроса (рассылают сообщения подписанным на них узлам), и посылает узлу консолидации управляющее сообщение после того, как все активные таймеры закончат оповещение о своем срабатывании. До получения этого сообщения узел консолидации запоминает в буфере все получаемые им сообщения с данными

мониторинга. К моменту получения сообщения об окончании срабатывания таймеров все датчики, которые должны были сгенерировать данные по сигналу активных таймеров, уже закончили это делать, и узел консолидации собирает накопленные данные в одно сообщение, а затем передает его дальше для обработки (например, для передачи по сети).

В момент старта система исполнения читает из конфигурационного файла список модулей, которые нужно загрузить. После загрузки модулей составляется список всех типов узлов, которые известны системе исполнения и будут доступны в процессе работы. Затем система читает из конфигурационного файла тип узла, который надо создать первым (стартовый узел). Этот узел создается и получает управляющее сообщение, в котором содержится номер файлового дескриптора, указывающего на оставшуюся часть конфигурационного файла. В дальнейшем стартовый узел читает оставшуюся часть конфигурационного файла и интерпретирует его по своему усмотрению. В частности, там могут содержаться команды для создания других узлов, связей между ними, таймеров, подписки на них узлов и задания прочей конфигурации. После окончания обработки стартовым узлом посланного ему управляющего сообщения система исполнения переходит в нормальный режим работы. В дальнейшем изменение конфигурации может производиться любым узлом. Это может быть реакцией на какие-то события, обнаруженные в данных мониторинга, либо же узел может принимать сетевые соединения и реагировать на полученные по сети команды. Эта функциональность не заложена в систему исполнения и должна реализовываться в узлах.

Таким образом, мы можем выстраивать произвольные конфигурации путей передачи информации мониторинга.

В качестве первого примера рассмотрим, как на основе нашей архитектуры будет выглядеть агент с конфигурацией, схожей по функциям с агентом существующих систем мониторинга. Его функцией будет получение данных и передача их в серверную часть. Конфигурация такого агента показана на рис. 2.

Он состоит из двух датчиков, узла консолидации и узла для передачи данных по сети. Кроме того, имеется таймер, который задает период сбора данных.

Другой вариант конфигурации, который недостижим при использовании существующих систем мониторинга, показан на рис. 3. В этой конфигурации показаны три цели обработки данных мониторинга.

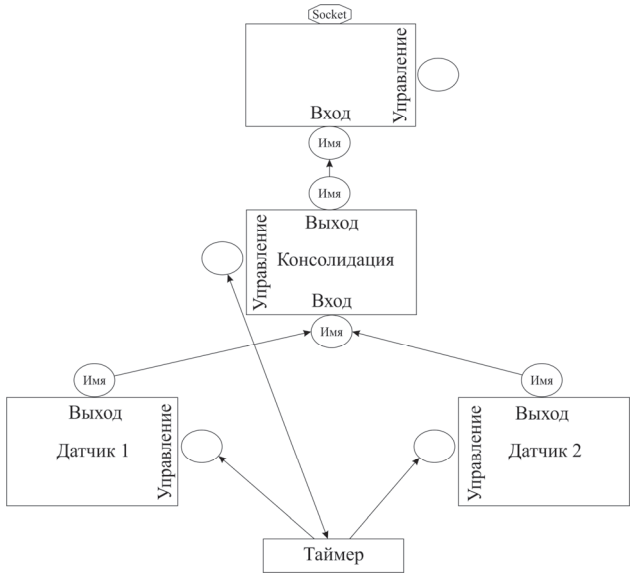

Рис. 2. Возможная конфигурация системы мониторинга

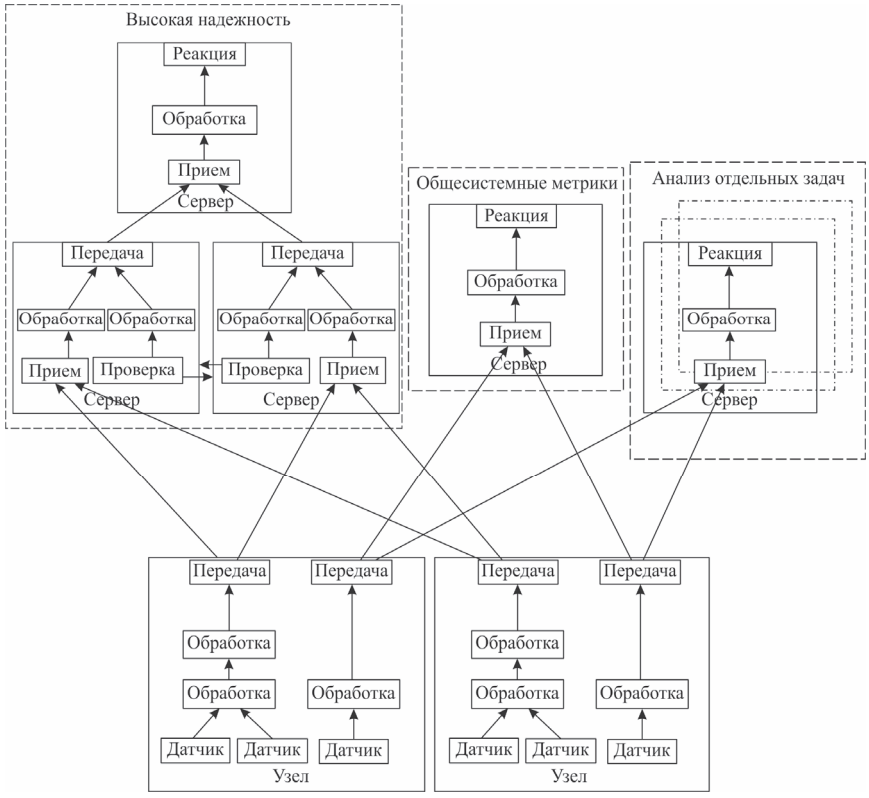

Рис. 3. Возможная конфигурация системы мониторинга (три цели обработки данных)

Первая цель требует высокой надежности обработки (это может быть отслеживание аварийных ситуаций). Реализующие ее агенты дублированы на двух независимых серверах, имеют узлы для слежения друг за другом. Агент, реализующий реакции, не дублирован, но может работать по сигналу от любого из дублей нижележащей части.

Вторая цель – построение общесистемных метрик. Для этого используются датчики, отличные от тех, что использованы в первой цели. Соответствующие данные идут на один выделенный сервер для обработки.

Наконец третья цель – анализ отдельных задач. Для этого запускаются отдельные агенты (или узлы внутри агентов) на каждую задачу, на которые поступают данные от вычислительных узлов, на которых запущена данная задача.

В настоящее время ведутся работы по реализации системы с указанной архитектурой. На основе уже реализованного прототипа были проведены оценки производительности. Была достигнута скорость передачи и приема данных в 350 тыс. пакетов в секунду, что составило примерно в 230 Мбит/с. В другом эксперименте проверялась возможность обрабатывать данные от многих источников, для чего осуществлялся подсчет средней загрузки процессора для примерно 5,5 тыс. вычислительных узлов. Скорость получения данных серверной частью составила примерно 47,5 тыс. значений в секунду, при этом загрузка процессора серверной частью составила менее 4 % одного ядра процессора Intel Xeon X5670, работающего на частоте 2,93 ГГц.

Работа выполняется при финансовой поддержке РФФИ, грант № 13-07-00775 А.

Библиографический список

1. Zenoss Community – Open Source Network Monitoring and Systems Management, available at: http://www.zenoss.org (дата обращения: 10.09.2014).

2. Zabbix, available at: http://www.zabbix.com (дата обращения: 10.09.2014).

3. Cacti® – The Complete RRDTool-based Graphing Solution, available at: http://www.cacti.net (дата обращения: 10.09.2014).

4. Massie M.L., Chun B.N., Culler D.E. The ganglia distributed monitoring system: design, implementation, and experience // Parallel Computing. – 2004. – Vol. 30, № 7. – P. 817–840.

5. The OpenNMS Project, available at: http://www.opennms.org (дата обращения: 10.09.2014).

6. Nagios – The Industry Standard in IT Infrastructure Monitoring, available at: http://www.nagios.org (дата обращения: 10.09.2014).

7. Collectd – The system statistics collection daemon, available at: https://collectd.org (дата обращения: 16.09.2014).

8. Gunter D. [et al.]. Dynamic monitoring of high-performance distributed applications // Proceedings 11th IEEE International Symposium on High Performance Distributed Computing: IEEE Comput. Soc. – 2002. – P. 163–170.

9. Mellor-Crummey J. [et al.]. HPCVIEW: A Tool for Top-down Analysis of Node Performance // The Journal of Supercomputing. – 2002. – Vol. 23, № 1. – P. 81–104.

10. Jagode H. [et al.]. A Holistic Approach for Performance Measurement and Analysis for Petascale Applications // Computational Science – ICCS 2009. Lecture Notes in Computer Science / ed. by G. Allen [et al.]. – Berlin; Heidelberg: Springer, 2009. – P. 686–695.

11. Alexander Z. [et al.]. Measurement and Dynamical Analysis of Computer Performance Data // Intelligent Data Analysis IX Lecture Notes in Computer Science / ed. by P.R. Cohen, N.M. Adams, M.R. Berthold. – Berlin; Heidelberg: Springer Berlin Heidelberg, 2010. – P. 18–29.

12. Adhianto L. [et al.]. HPCTOOLKIT: tools for performance analysis of optimized parallel programs // Concurrency and Computation: Practice and Experience. – 2009. – Vol. 22, № 6.

13. Eisenhauer G. [et al.]. Falcon: on-line monitoring and steering of large-scale parallel programs // Proceedings Frontiers'95. The Fifth Symposium on the Frontiers of Massively Parallel Computation. – McLean, VA: IEEE Comput. Soc. Press, 1995. – P. 422–429.

14. Kluge M., Hackenberg D., Nagel W.E. Collecting Distributed Performance Data with Dataheap: Generating and Exploiting a Holistic System View // Procedia Computer Science. – 2012. – Vol. 9. – P. 1969–1978.

15. Mooney R., Schmidt K.P., Studham R.S. NWPerf: a system wide performance monitoring tool for large Linux clusters // IEEE International Conference on Cluster Computing (IEEE Cat. No. 04EX935). – 2004. – P. 379–389.

16. Ries B. [et al.]. The paragon performance monitoring environment // Supercomputing'93, Proceedings: IEEE. – 1993. – P. 850–859.

References

1. Zenoss Community – Open Source Network Monitoring and Systems Management, available at: http://www.zenoss.org (accessed 10 September 2014).

2. Zabbix, available at: http://www.zabbix.com (accessed 10 September 2014).

3. Cacti® – The Complete RRDTool-based Graphing Solution, available at: http://www.cacti.net (accessed 10 September 2014).

4. Massie M.L., Chun B.N., Culler D.E. The ganglia distributed monitoring system: design, implementation, and experience. *Parallel Computing*, 2004, vol. 30, no. 7, pp. 817-840.

5. The OpenNMS Project, available at: http://www.opennms.org (accessed 10 September 2014).

6. Nagios – The Industry Standard in IT Infrastructure Monitoring, available at: http://www.nagios.org (accessed 10 September 2014).

7. Collectd – The system statistics collection daemon, available at: https://collectd.org (accessed 16 September 2014).

8. Gunter D. [et al.]. Dynamic monitoring of high-performance distributed applications. *Proceedings 11th IEEE International Symposium on High Performance Distributed Computing. IEEE Comput. Soc*., 2002, pp. 163-170.

9. Mellor-Crummey J. [et al.]. HPCVIEW: A Tool for Top-down Analysis of Node Performance. *The Journal of Supercomputing*, 2002, vol. 23, no. 1, pp. 81-104.

10. Jagode H. [et al.]. A Holistic Approach for Performance Measurement and Analysis for Petascale Applications. *Computational Science – ICCS 2009. Lecture Notes in Computer Science.* Ed. by G. Allen [et al.]. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 686-695.

11. Alexander Z. [et al.]. Measurement and Dynamical Analysis of Computer Performance Data. *Intelligent Data Analysis IX Lecture Notes in Computer Science.* Ed. by P.R. Cohen, N.M. Adams, M.R. Berthold. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 18-29.

12. Adhianto L. [et al.]. HPCTOOLKIT: tools for performance analysis of optimized parallel programs. *Concurrency and Computation: Practice and Experience*, 2009, vol. 22, no. 6.

13. Eisenhauer G. [et al.]. Falcon: on-line monitoring and steering of large-scale parallel programs. *Proceedings Frontiers'95. The Fifth Symposium on the Frontiers of Massively Parallel Computation.* McLean, VA: IEEE Comput. Soc. Press, 1995, pp. 422-429.

14. Kluge M., Hackenberg D., Nagel W.E. Collecting Distributed Performance Data with Dataheap: Generating and Exploiting a Holistic System View. *Procedia Computer Science*, 2012, vol. 9, pp. 1969-1978.

15. Mooney R., Schmidt K.P., Studham R.S. NWPerf: a system wide performance monitoring tool for large Linux clusters. *2004 IEEE International Conference on Cluster Computing (IEEE Cat. No. 04EX935)*, 2004, pp. 379-389.

16. Ries B. [et al.]. The paragon performance monitoring environment. *Supercomputing'93. Proceedings IEEE*, 1993, pp. 850-859.

Об авторе

Стефанов Константин Сергеевич (Москва, Россия) – кандидат физико-математических наук, старший научный сотрудник Научно-исследовательского вычислительного центра МГУ им. М.В. Ломоносова (119234, г. Москва, Ленинские Горы, 1, стр. 4, e-mail: cstef@parallel.ru).

About the author

Konstantin S. Stefanov (Moscow, Russian Federation) – Ph. D. in Physics and Mathematical Sciences, Senior Research Fellow, Research Computing Center, M.V. Lomonosov Moscow State University (1, building 4, Leninskiye Gory, Moscow, 119234, Russian Federation, e-mail: cstef@parallel.ru).

Получено 1.10.2014