



Обзор и сравнение современных систем мониторинга производительности в HPC

Международная конференция
«Суперкомпьютерные дни в России»

Паокин Андрей Викторович

МГУ имени М.В. Ломоносова
Научно-исследовательский вычислительный центр
Лаборатория параллельных информационных технологий

andrejpaokin@yandex.ru

28 сентября 2026 г.

Ключевая задача любой системы мониторинга — непрерывный контроль и диагностика состояния аппаратного обеспечения. Суперкомпьютеры – дорогой вычислительный ресурс, находящийся в коллективном пользовании, поэтому у системы мониторинга в НРС есть дополнительные функции:

1. Сбор данных о **производительности заданий** пользователей
2. Анализ собранных данных:
 - 2.1 Визуализация
 - 2.2 Обнаружение узких мест производительности в заданиях пользователей с помощью эвристических критериев и методов машинного обучения.
3. Реакция на анализ:
 - 3.1 Информирование обычных пользователей и администраторов
 - 3.2 Снятие неэффективных заданий



- ▶ Узловой агент – фоновый процесс, работающий на узле и собирающий данные об этом узле. В режиме push узловой агент сам инициирует передачу в серверную часть системы мониторинга или вышестоящее промежуточное звено, в режиме pull – слушает и отдаёт данные по запросу внешнего компонента.
- ▶ Серверная часть, отвечающая за хранение данных. Необязательно создана разработчиками системы мониторинга, может быть просто базой данных (InfluxDB, Prometheus).
- ▶ Компонент для интеграции с планировщиком заданий (например, epilog-скрипт).
- ▶ Компонент доступа пользователей и администраторов к данным (в т.ч. визуализация, отчёты в терминале).



Компоненты для обеспечения масштабируемости, отказоустойчивости:

- ▶ Брокер сообщений принимает данные, даже если центральный сервер не отвечает и перегружен.
- ▶ Промежуточные звенья между узловыми агентами и центральным сервером.
- ▶ Балансировщик нагрузки.

Другие компоненты:

- ▶ Компонент анализа собранных данных (выявление приложений с нулевой загрузкой GPU, дисбалансом загрузки ядер и др.).



В СКЦ МГУ давно осуществляется мониторинг производительности приложений пользователей (суперкомпьютеры «Ломоносов» и «Ломоносов-2») с помощью собственных разработок:

- ▶ Узловой агент **DiMMon** каждую секунду формирует данные о работе CPU, GPU, памяти, сетей хранения и межузлового обмена и каждую минуту посылает агрегированные (мин, макс, среднее, медиана, ...) значения.
- ▶ **TASC** принимает, анализирует и записывает данные в долгосрочные хранилища с разной степенью детализации и сроком хранения.
- ▶ Модуль системы поддержки функционирования СКЦ **Octoshell** осуществляет доступ пользователей к собранным данным.

Также **Redash** используется администраторами для визуализации данных.

DiMMon работает без нареканий, однако у нас сейчас нет отдельного разработчика для его поддержки. Можно ли его **заменить на другой узловой агент**?



1. **ClusterCockpit**

Немецкие СКЦ: Erlangen National High Performance Computing Center, Paderborn Center for Parallel Computing, German Climate Computing Center, ...

Helma TOP500#73

2. **LDMS (OVIS)**

El Capitan TOP500#2, Perlmutter TOP500#34

3. **Jobstats**

Небольшие американские СКЦ

4. **HPCPerf Stats** (ранее TACC Stats)

Frontera TOP500#90

Все системы с открытым исходным кодом и у всех последний коммит был в этом месяце.



Рассматриваемые системы мониторинга собирают разный набор данных, иногда недостаточный, особенно у CPU. Метрики CPU:

1. Jobstats: время работы ядра в пользовательском и привилегированном режиме, загрузка CPU.
2. ClusterCockpit: настраиваемый набор через *Likwid*.
3. LDMS: настраиваемый набор через *PAPI*.
4. HPCPerf Stats: настраиваемый набор через *perf_event*.

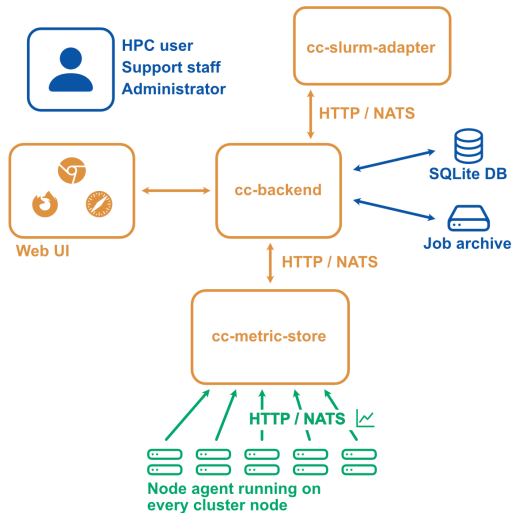
Количество специализированных регистров CPU (Performance Monitoring Counters) для сбора низкоуровневых событий ограничено 4–8. Для сбора значительного числа событий необходима ротация счётчиков или мультиплексирование.

Likwid, PAPI, perf_event упрощают данную задачу.

Вывод: уже на этой стадии администраторы СКЦ могут отсеять неподходящие системы мониторинга. Упоминание в документации одного из инструментов – признак широких возможностей по сбору метрик.



ClusterCockpit: схема компонентов



1. **cc-metric-collector** – узловой агент, состоящий из 4 компонент:
 - ▶ collectors: сборщики «сырых» данных, каждый коллектор отвечает за свою область: CPU (Likwid), memstat, io и др.
 - ▶ sinks: «отправители» данных (InfluxDB, Prometheus, Ganglia, просто по HTTP, stdout и др.)
 - ▶ receivers получают дополнительные сведения из внешних хранилищ.
 - ▶ router: предобработка данных на узле (фильтрация, агрегация и др.)
2. **cc-backend** – пользовательское веб-приложение и бекенд. Хранит метаданные задания в реляционной СУБД SQLite3.
3. **cc-slurm-adapter** – работает на том же узле, где и главный демон SLURM *slurmctld*. Передаёт метаданные заданий в cc-backend.
4. **cc-metric-store** (внутри cc-backend или внешний) – хранилище метрик текущих и недавних заданий в ОЗУ.
5. **Архив заданий** – база SQLite3 для быстрого поиска и файловое хранилище всех сведений о задании в форматах JSON, Parquet, S3-совместимом.
6. Транспорт: **HTTP** или **брокер сообщений NATS**.



- ✓ Самая понятная документация (развертывание и настройка узлового агента за 5 минут).
- ✓ Модульный, легко модифицируемый, компоненты отчуждаемы (узловой агент, центральный сервер).
- ✓ Простое добавление новых метрик (можно без программирования).
- ✓ Предобработка данных прямо на узловом агенте.
- ✗ Ошибки в компонентах, которые сейчас не используются в реальных установках: агрегация, HTTP-сервер для pull-режима Prometheus, однако они были легко исправлены.
- ✗ Долгий поиск в архиве заданий по неиндексируемым метрикам.



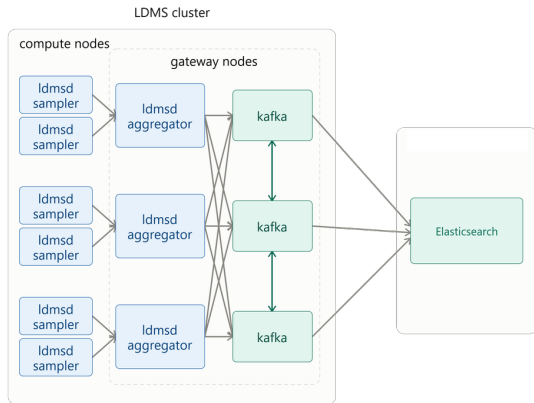


Рис. 1: Поток данных LDMS на El Capitan

Компоненты:

1. Сэмплеры – узловые агенты.
2. Агрегаторы собирают данные с сэмплеров и нижестоящих агрегаторов. Передают данные в хранилище или в следующие агрегаторы в иерархии.
3. Maestro – балансировщик нагрузки (если нужен).
4. Транспорт: сокеты и remote direct memory access.

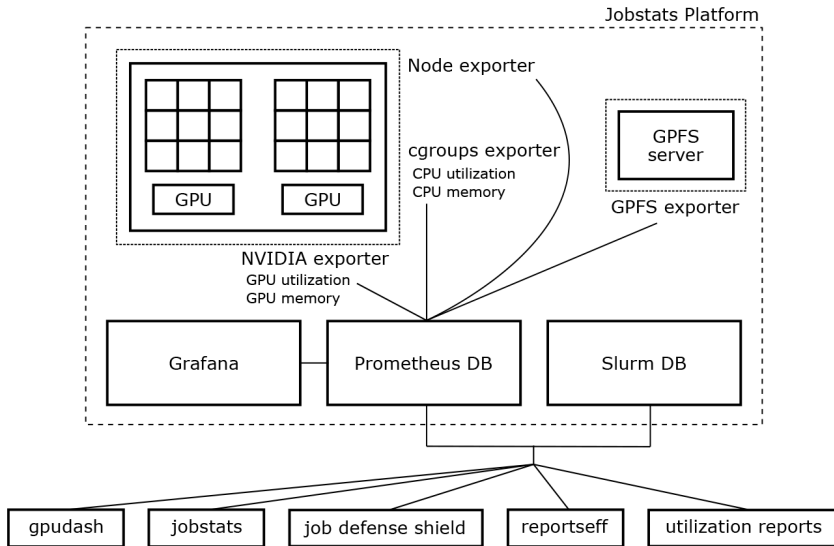
Как сэмплеры, так и агрегаторы реализованы через один и тот же демон *ldmsd*. *ldmsd* имеет **плагинную** архитектуру:

- ▶ Плагины сбора данных: CPU (PAPI), GPU, память, сети.
- ▶ Плагины хранения: CSV, Kafka, InfluxDB, SOS (Scalable Object Store), MySQL, RabbitMQ.



- ✓ Успешно работает много лет на крупных системах и продолжает развиваться (в статье 2023 года описываются новые возможности, в т.ч. в плане производительности).
- ✓ Много материалов: документация, презентации, видео, отдельная конференция LDMScon.
- ✓ Можно расширить плагинами.
- ✗ Усложнённый старт.
- ✗ Может быть избыточным для небольших установок.
- ✗ Нет фильтрации и агрегации на узловых агентах и агрегаторах (только через сторонние плагины).
- ▶ Система не привязана к какому-либо хранилищу, хотя авторы проекта разработали специальное хранилище SOS для LDMS.





- ▶ Система сбора данных практически полностью основана на стандартных инструментах. Prometheus DB опрашивает экспортеры на узлах.
- ▶ Используется в нескольких небольших американских СКЦ.
- ✗ Для каждого вида ресурсов (CPU, GPU, память, ввод-вывод) необходим **отдельный экспортер** на узле (процесс ОС), работающий по HTTP.
- ✗ Ограниченное количество метрик.
- ✓ Пользователю доступны сведения о работе задания через веб- и терминальный интерфейсы.
- ✓ Возможность снятия заданий с нулевой утилизацией ресурсов.



Архитектура:

1. **monitor** – узловой агент сбора данных о CPU, GPU, памяти, сетях и вводе-выводе.
2. **hpcperfstats** собирает данные с узловых агентов и сохраняет в PostgreSQL или TimescaleDB.
3. Веб-приложение для просмотра метрик (Django / кэширование через Redis).
4. Транспорт: брокер сообщений RabbitMQ.

Свойства:

- ▶ Фиксированный стек.
- ✗ Нет руководства по расширению функциональности.



При выборе системы мониторинга:

- ▶ надо знать, какие данные система мониторинга может собирать, особое внимание уделить CPU, GPU и ФС.
- ▶ иногда необходимо искать баланс между простотой управления и масштабируемостью/отказоустойчивостью.

Расширяемые и хорошо документированные системы мониторинга позволяют организовать сбор дополнительных метрик.

Подходящим вариантом может оказаться создание *собственной системы мониторинга* с использованием готовых компонент других систем.

