Технологии параллельной обработки графов: проблемы, подходы, перспективы

Александр Фролов

Лаборатория DISLab, ОАО «НИЦЭВТ»

GraphHPC-2014, Москва
5 марта 2014 г.

План доклада

- Проблемы анализа больших графов
- Классификация задач анализа графов
- Аппаратные платформы для решения графовых задач
- Модели и средства параллельной обработки графов
- Предварительные результаты сравнительного оценочного тестирования нескольких систем параллельной обработки графов на задаче SSSP
- Выводы

Откуда возникают большие графы?

- Интернет (WWW)
  - На февраль 2014 – 20 миллиардов страниц[1]
  - По оценке Google – более 1 триллиона
- Социальные медиа
  - Блогосфера: 2011 – 172×10^6 ($+10^6$/день), 2013 – неизвестно
  - Facebook: 2010 – 500×10^6, 2013 – 1.1×10^9 (650×10^6 акт.польз./день), 140×10^9 связей
  - LinkedIn: 2013 – 8×10^6, 60×10^6 связей
  - Twitter: 2011 – 140×10^6 сообщений/день
- Транспортные сети
- Биоинформатика

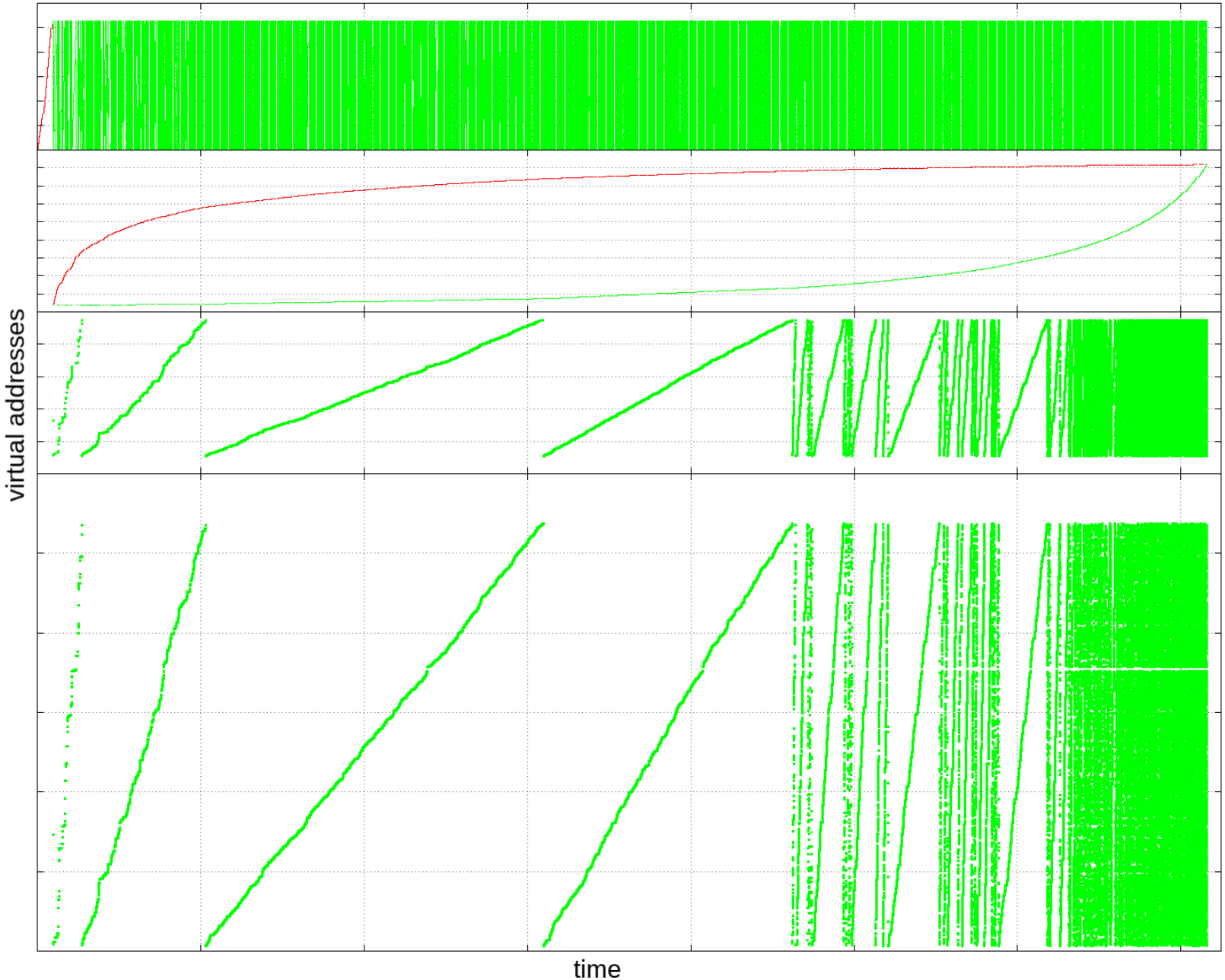
[1] http://www.worldwidewebsize.com

Проблемы анализа больших графов [2]

- **Data-driven computations**: Зависимость вычислений от данных (топологии графа). Невозможность применения методов статического распараллеливания вычислений.
- **Unstructured problems & poor locality**: Работа с нерегулярными, неструктурированными данными. Низкая пространственно-временная локализация обращений к памяти.
- **High data access to computation ratio**: Преобладание команд доступа к памяти над командами выполнения арифметических операций.
- **Проблема декомпозиции графа**: Большинство графов реального мира имеют степенной закон распределения степени вершин ($P(k) \sim k^{-a}$).
  - *Пример: DBPedia около 90% вершин имеют менее чем 5 соседей, есть несколько вершин, которые имеют более чем 100000 соседей.*

[2] Адаптировано из Andrew Lumsdaine, Douglas Gregor, Bruce Hendrickson, Jonathan W. Berry: Challenges in Parallel Graph Processing. Parallel Processing Letters 17(1): 5-20 (2007)

Профиль работы с памятью (данными) теста Graph500



Классификация задач анализа графов

- **По типу графов**
  - статические графы (static graph analysis)
  - динамические графы (dynamic graph analysis)
  - обработка потоков вершин и ребер (streaming graph analysis)
- **По типу обработки**
  - в режиме реального времени (online)
    - Обход графа (напр., BFS), поиска кратчайших путей (напр., SSSP), поиск максимальных клик, поиск подграфов по заданному шаблону, выполнение запросов к графу (SQL-like) и др.
  - в режиме выполнения заданий (offline, batch processing)
    - PageRank, оценка диаметра, поиск связных компонент графа, раскраска вершин графа, кластеризация гафа и др.

Аппаратные платформы

- **По архитектуре вычислительного узла**
  - Многоядерные, однородные узлы
    - SMP-узлы (1x,2x,4x сокетные узлы)
  - Многоядерные, неоднородные (гетерогенные) узлы
    - SMP-узлы + ускорители (GPU, MIC, FPGA)
  - Специализированные узлы на базе ПЛИС
- **По архитектуре памяти**
  - SMP-системы с NUMA-памятью
    - SMP-узлы (1x,2x,4x сокетные узлы)
  - Системы с ccNUMA-памятью
    - SGI UV, NumaScale
  - Системы с распределенной памятью
    - Infiniband-кластеры, Ethernet-кластеры, Custom Interconnect-кластеры
  - Системы с общей NUMA-памятью
    - Cray XMT2 (Yarcdata uRiKA)

Программные модели и средства

- **RDBMS**
  - Cassandra, SAP HANA и др.
- **MapReduce**
  - Generic MR: Hadoop, Yarn, Dryad, Stratosphere, Haloop
  - Graph-optimized MR: Pegasus, Surfer, GBASE, GraphX, Spark
- **Vertex-centric/BSP**
  - Pregel (Giraph, Hama, Mizan и т.д.), Parallel BGL
- **Vertex-centric/Data,Message-driven**
  - GraphLab, SWARM, Trinity и т.д.
- **Fine-grained Threaded Shared Memory/PGAS**
  - GraphCT, STINGER, Grappa
- **Специализированные языки программирования**
  - Проблемно-ориентированные языки программирования (DSL)
    - Green-Marl, Exedra
  - Языки обработки данных
    - Datalog, R и др.
  - Языки запросов к графовым СУБД
    - SPARQL, G-SPARQL, Cypher (Neo4j) и т.д.

Выбор фреймворков для исследования на тесте SSSP

Фреймворк	**Тип**	**Модель**	**Платформа**	**Реализация**
Giraph	Библиотека	Vertex-centric/BSP	Распределенные системы, cloud computing	Java, Hadoop
GraphLab	Библиотека	Vertex-centric/GAS	Распределенные системы, cloud computing	C++, MPI, TCP/IP sockets
Yappi	Библиотека	Vertex-centric/BSP	Кластерные системы, супер-компьютеры	C++, MPI, Shmem
Green-Marl	DSL	Thread/Data parallel (OMP-style)	SMP-системы, распределенные системы, cloud computing	DSL compiler, C++/OpenMP, Giraph, GPS

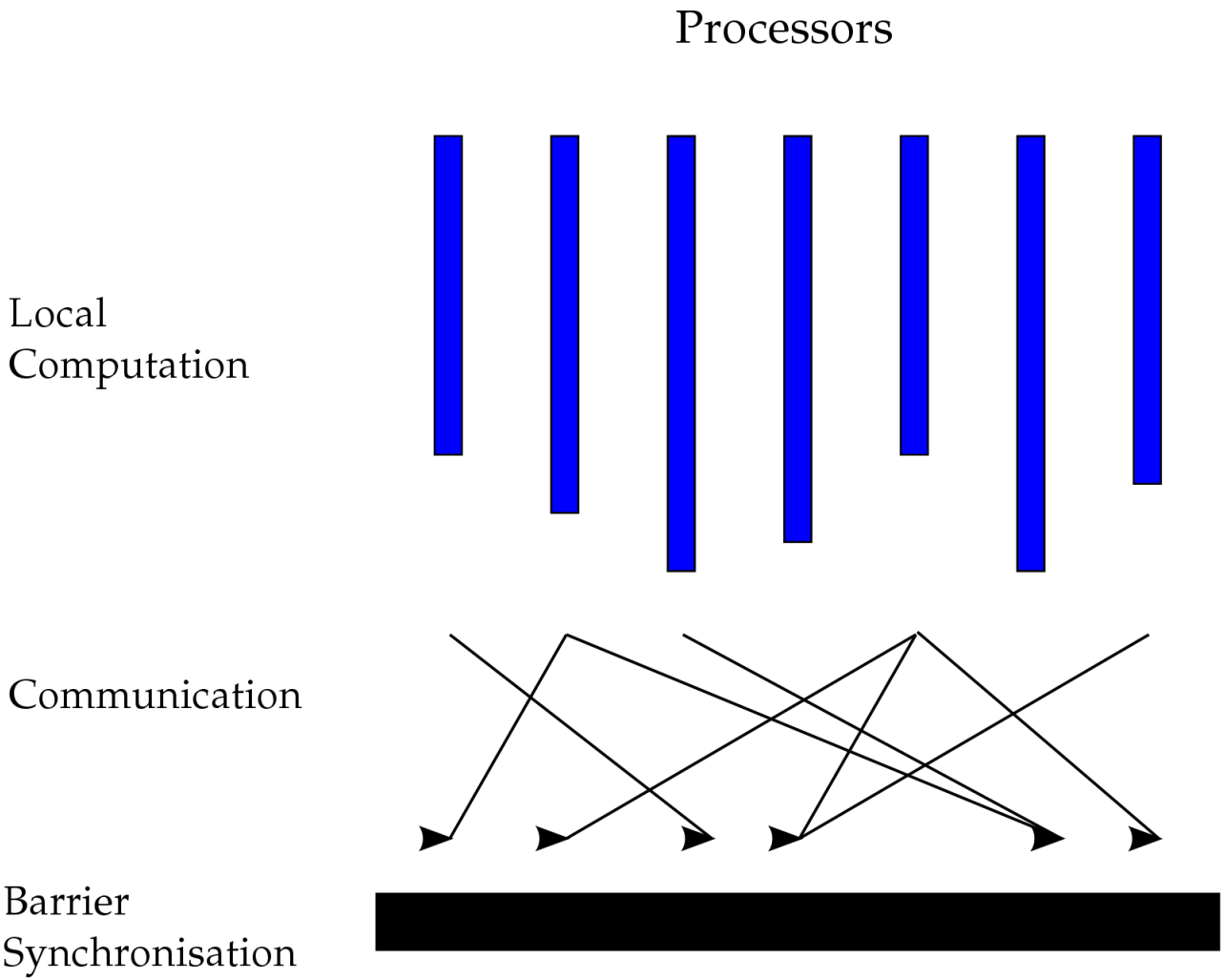
Pregel

- Разработана в Google в 2010 [3]
- Программная модель – *Vertex-centric + Bulk Synchronous Parallel*
- Коммуникации с помощью обмена сообщениями между вершинами
- Разработано множество исследовательских проектов на основе Pregel за последние 3 года [4]
- Плюсы Pregel:
  - отсутствие дедлоков (deadlock-free) и "гонок к данным" (data races-free)
  - мелкозернистый параллелизм (на уровне обработки вершин)
  - агрегация коротких сообщений
  - отказоустойчивость (естественный способ снятия контрольных точек)
- Минусы Pregel:
  - производительность (BSP требует глобального барьера после каждого супершага, выделение вершин в виде объектов (C++) требует много памяти)

[3] Grzegorz Malewicz et al. Pregel: a system for large-scale graph processing. In Proceedings of the 2010 ACM SIGMOD International Conference on Management of data (SIGMOD '10). ACM, New York, NY, USA, 135-146.

[4] Apache Giraph, Apache Hama, GPS, GraphLab, Signal/Collect, Phoebus, GoldenOrb, HipG, Mizan

Bulk Synchronous Parallel (BSP) [5]



Источник: http://en.wikipedia.org/wiki/Bulk_synchronous_parallel

[5] **Leslie G. Valiant. 1990. A bridging model for parallel computation. Commun. ACM 33, 8 (August 1990), 103-111.**

Pregel (рисунки для понимания) [6]

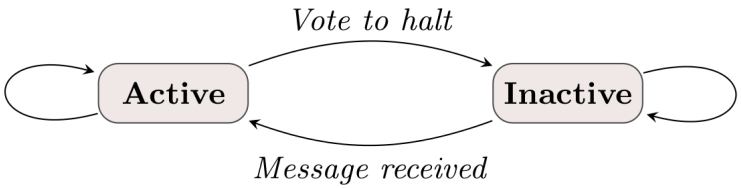


Figure 1: Vertex State Machine

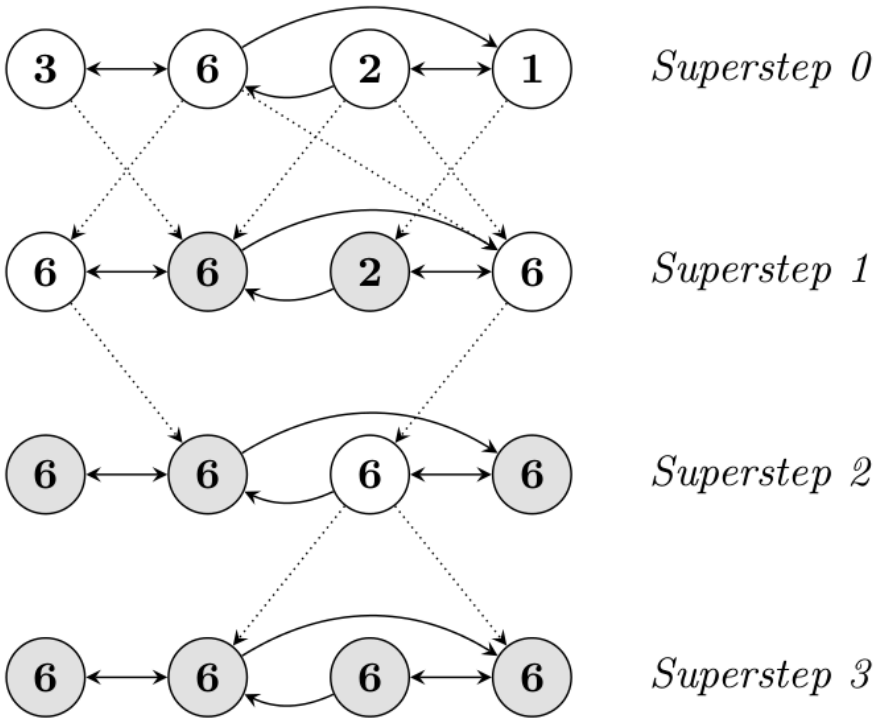
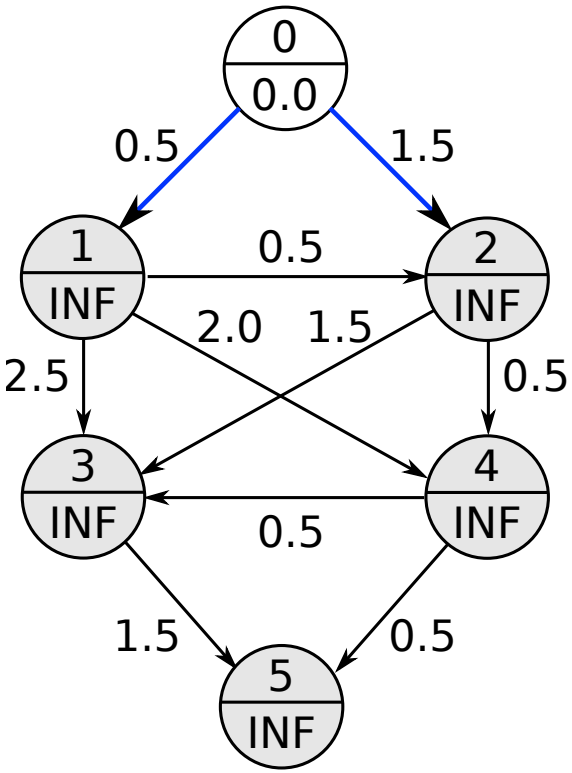


Figure 2: Maximum Value Example. Dotted lines are messages. Shaded vertices have voted to halt.

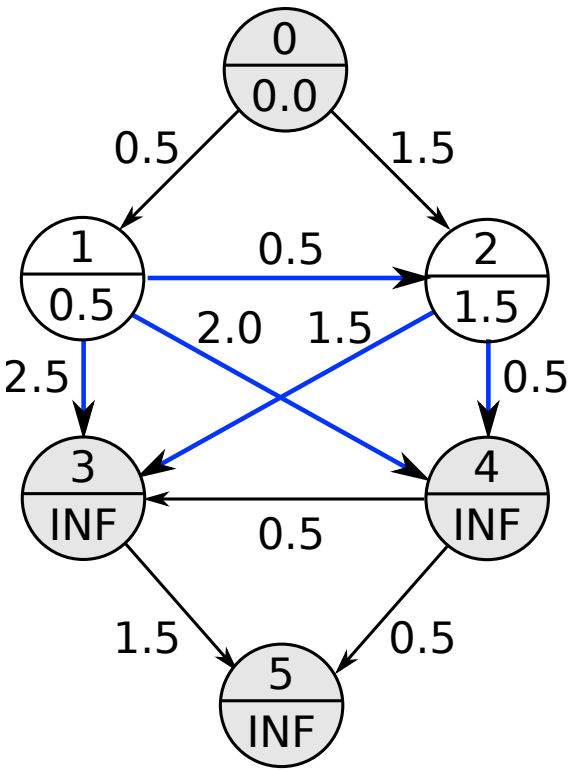
[6] Grzegorz Malewicz et al. Pregel: a system for large-scale graph processing. In Proceedings of the 2010 ACM SIGMOD International Conference on Management of data (SIGMOD '10). ACM, New York, NY, USA, 135-146.

Реализация SSSP на Pregel (пример выполнения)



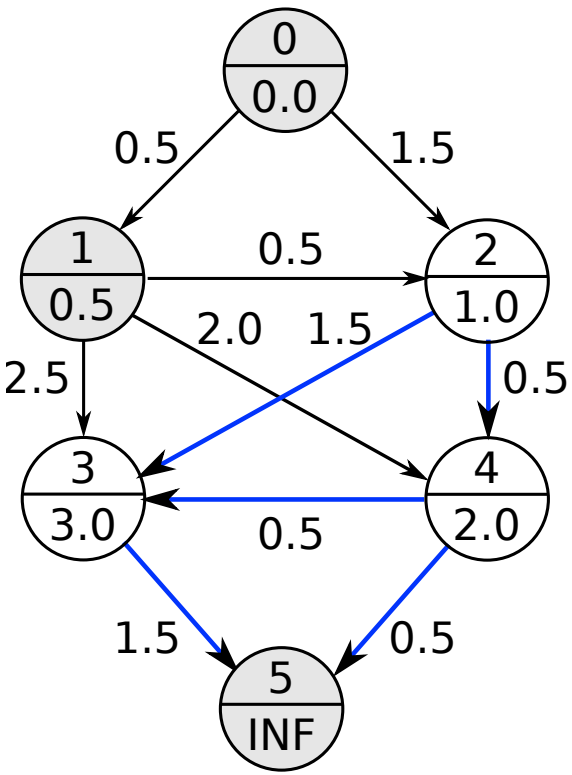

```
1  class ShortestPathVertex
2    : public Vertex<int, int, int> {
3      void Compute(MessageIterator *msgs) {
4        int mindist = IsSource(vertex_id()) ? 0 : INF;
5        for (; !msgs->Done(); msgs->Next())
6          mindist = min(mindist, msgs->Value());
7        if (mindist < GetValue()) {
8          *MutableValue() = mindist;
9          OutEdgeIterator iter = GetOutEdgeIterator();
10         for (; !iter.Done(); iter.Next())
11           SendMessageTo(iter.Target(),
12                 mindist + iter.GetValue());
13       }
14       VoteToHalt();
15     }
16 };
```

Реализация SSSP на Pregel (пример выполнения)



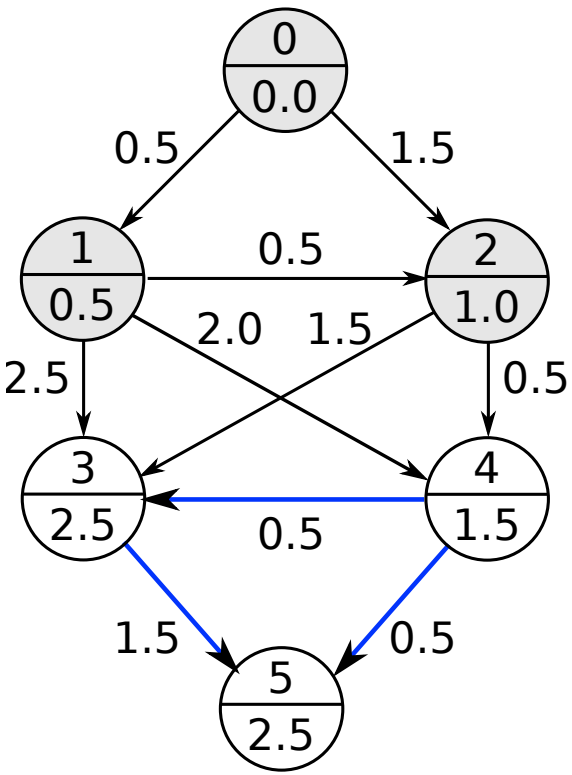

```
1  class ShortestPathVertex
2    : public Vertex<int, int, int> {
3      void Compute(MessageIterator *msgs) {
4        int mindist = IsSource(vertex_id()) ? 0 : INF;
5        for (; !msgs->Done(); msgs->Next())
6          mindist = min(mindist, msgs->Value());
7        if (mindist < GetValue()) {
8          *MutableValue() = mindist;
9          OutEdgeIterator iter = GetOutEdgeIterator();
10         for (; !iter.Done(); iter.Next())
11           SendMessageTo(iter.Target(),
12                 mindist + iter.GetValue());
13       }
14       VoteToHalt();
15     }
16 };
```

Реализация SSSP на Pregel (пример выполнения)



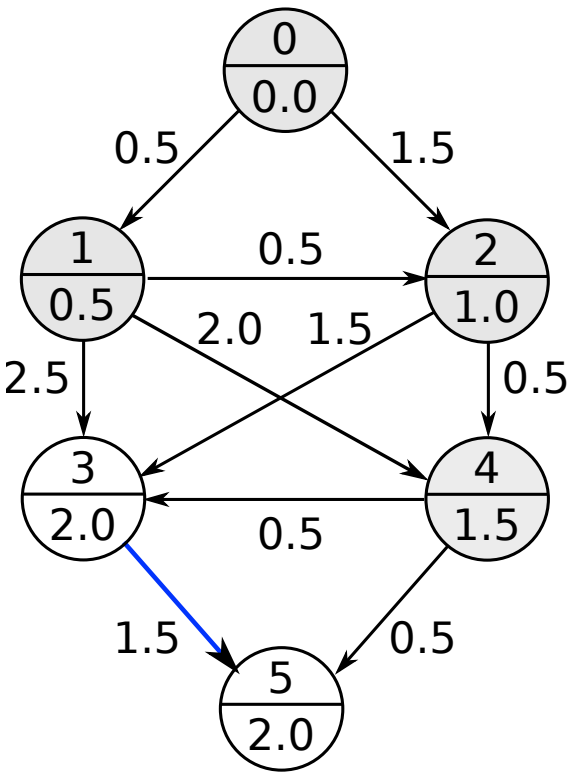

```
1  class ShortestPathVertex
2    : public Vertex<int, int, int> {
3      void Compute(MessageIterator *msgs) {
4        int mindist = IsSource(vertex_id()) ? 0 : INF;
5        for (; !msgs->Done(); msgs->Next())
6          mindist = min(mindist, msgs->Value());
7        if (mindist < GetValue()) {
8          *MutableValue() = mindist;
9          OutEdgeIterator iter = GetOutEdgeIterator();
10         for (; !iter.Done(); iter.Next())
11           SendMessageTo(iter.Target(),
12                 mindist + iter.GetValue());
13       }
14       VoteToHalt();
15     }
16 };
```

Реализация SSSP на Pregel (пример выполнения)



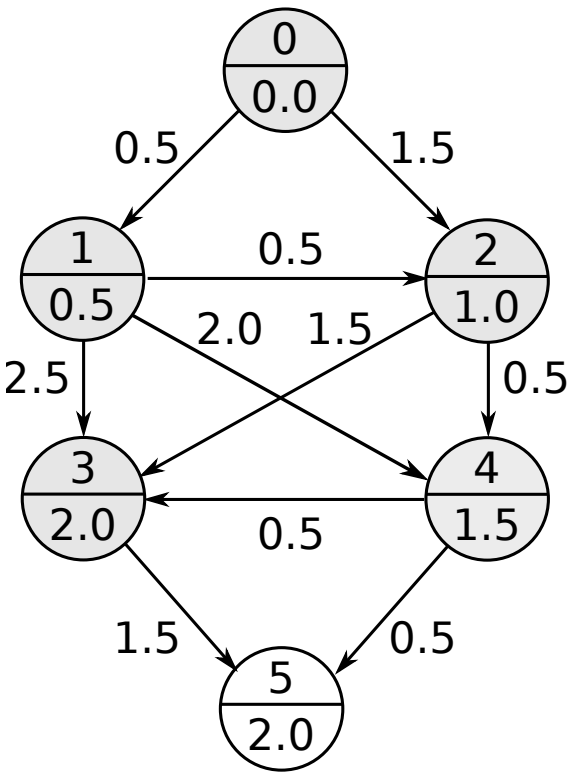

```
1  class ShortestPathVertex
2    : public Vertex<int, int, int> {
3      void Compute(MessageIterator *msgs) {
4        int mindist = IsSource(vertex_id()) ? 0 : INF;
5        for (; !msgs->Done(); msgs->Next())
6          mindist = min(mindist, msgs->Value());
7        if (mindist < GetValue()) {
8          *MutableValue() = mindist;
9          OutEdgeIterator iter = GetOutEdgeIterator();
10         for (; !iter.Done(); iter.Next())
11           SendMessageTo(iter.Target(),
12                 mindist + iter.GetValue());
13       }
14       VoteToHalt();
15     }
16 };
```

Реализация SSSP на Pregel (пример выполнения)



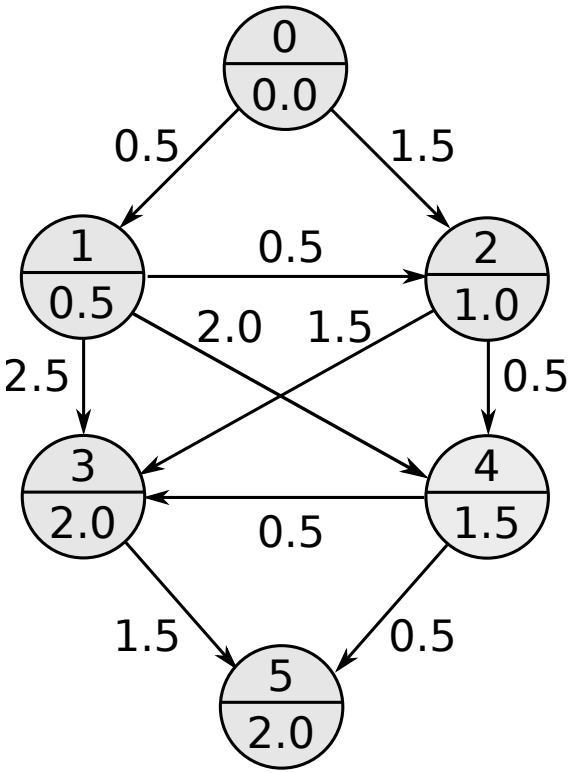

```
1  class ShortestPathVertex
2    : public Vertex<int, int, int> {
3      void Compute(MessageIterator *msgs) {
4        int mindist = IsSource(vertex_id()) ? 0 : INF;
5        for (; !msgs->Done(); msgs->Next())
6          mindist = min(mindist, msgs->Value());
7        if (mindist < GetValue()) {
8          *MutableValue() = mindist;
9          OutEdgeIterator iter = GetOutEdgeIterator();
10         for (; !iter.Done(); iter.Next())
11           SendMessageTo(iter.Target(),
12                 mindist + iter.GetValue());
13       }
14       VoteToHalt();
15     }
16 };
```

Реализация SSSP на Pregel (пример выполнения)




```
1  class ShortestPathVertex
2    : public Vertex<int, int, int> {
3      void Compute(MessageIterator *msgs) {
4        int mindist = IsSource(vertex_id()) ? 0 : INF;
5        for (; !msgs->Done(); msgs->Next())
6          mindist = min(mindist, msgs->Value());
7        if (mindist < GetValue()) {
8          *MutableValue() = mindist;
9          OutEdgeIterator iter = GetOutEdgeIterator();
10         for (; !iter.Done(); iter.Next())
11           SendMessageTo(iter.Target(),
12                 mindist + iter.GetValue());
13       }
14       VoteToHalt();
15     }
16 };
```

Реализация SSSP на Pregel (пример выполнения)




```
1  class ShortestPathVertex
2    : public Vertex<int, int, int> {
3      void Compute(MessageIterator *msgs) {
4        int mindist = IsSource(vertex_id()) ? 0 : INF;
5        for (; !msgs->Done(); msgs->Next())
6          mindist = min(mindist, msgs->Value());
7        if (mindist < GetValue()) {
8          *MutableValue() = mindist;
9          OutEdgeIterator iter = GetOutEdgeIterator();
10         for (; !iter.Done(); iter.Next())
11           SendMessageTo(iter.Target(),
12                 mindist + iter.GetValue());
13       }
14       VoteToHalt();
15     }
16 };
```

Apache Giraph

- Наиболее популярный open-source клон Google Pregel
- Разрабатывается с 2012 года
- Java + Hadoop/Yarn (map only) + (HDFS + Zookeper)
- Поддержка мультитредининга внутри узла (multithreaded mappers)
- Коммуникации – TCP/IP (Netty)
- Распределение вершин: $Hash(Vertex_{id}) \mod N$
- Используется в Facebook (с лета 2013), LinkedIn, Yahoo!
- Вычисления в памяти (in-memory)
- Удобные средства отладки и мониторинга (JobTracker, TaskTracker), возможность работать через http

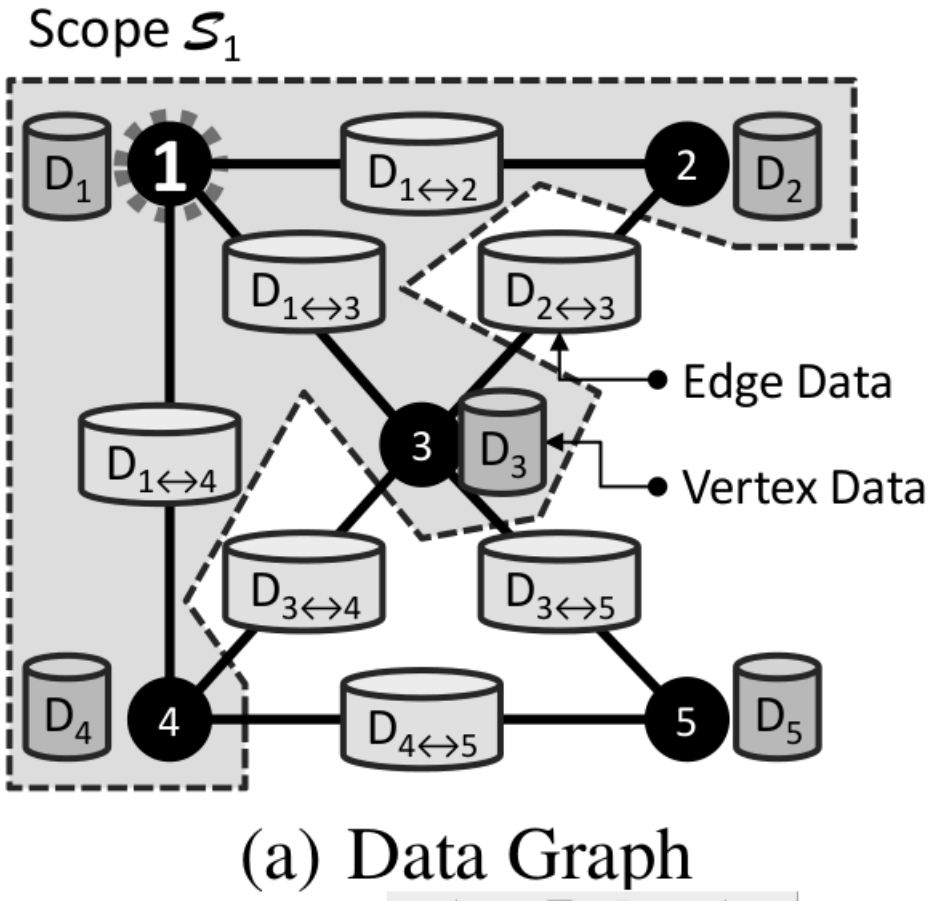
Yappi [Yet Another Pregel Implementation]

- DISLab/НИЦЭВТ, начало разработки – лето 2012
- Исследовательский проект
  - Цель #1: портирование модели Pregel на кластер с высокоскоростной коммуникационной сетью
  - Цель #2: отработка принципов создания runtime-системы с мелкозернистым параллелизмом
- C++, Pthreads, MPI/Shmem
- Базовая функциональность Pregel
- Масштабируемость: worker/comm-треды (ядра), MPI-процессы (узлы)

GraphLab

- Разрабатывается с 2009 года в Carnegie Mellon University
- Модель Vertex-centric, Gather–Apply–Scatter (GAS)
- Поддержка разных режимов выполнения программы Asynchronous, Synchronous (BSP)
- Поддержка трех видов консистентности памяти (vertex consistency, edge consistency, full consistency)
- Поддержка различных типов распределения вершин по узлам системы
  - random, grid, oblivious (heuristic)
- Коммуникации TCP/IP sockets
- Вычисления в памяти (in-memory)
- Поддержка POSIX FS и HDFS

GraphLab: Gather – Apply – Scatter (GAS) [7]



- **Gather:** $\Sigma \leftarrow \bigoplus_{v \in Nbrs[u]} g(D_u, D_{(u,v)}, D_v)$
- **Apply:** $D_u^{new} \leftarrow a(D_u, \Sigma)$
- **Scatter:** $\forall v \in Nbrs[u] : (D_{(u,v)}) \leftarrow (D_u^{new}, D_{(u,v)}, D_v)$

[7] Joseph E. Gonzalez, Yucheng Low, Haijie Gu, Danny Bickson, and Carlos Guestrin. 2012. PowerGraph: distributed graph-parallel computation on natural graphs. In Proceedings of the 10th USENIX conference on Operating Systems Design

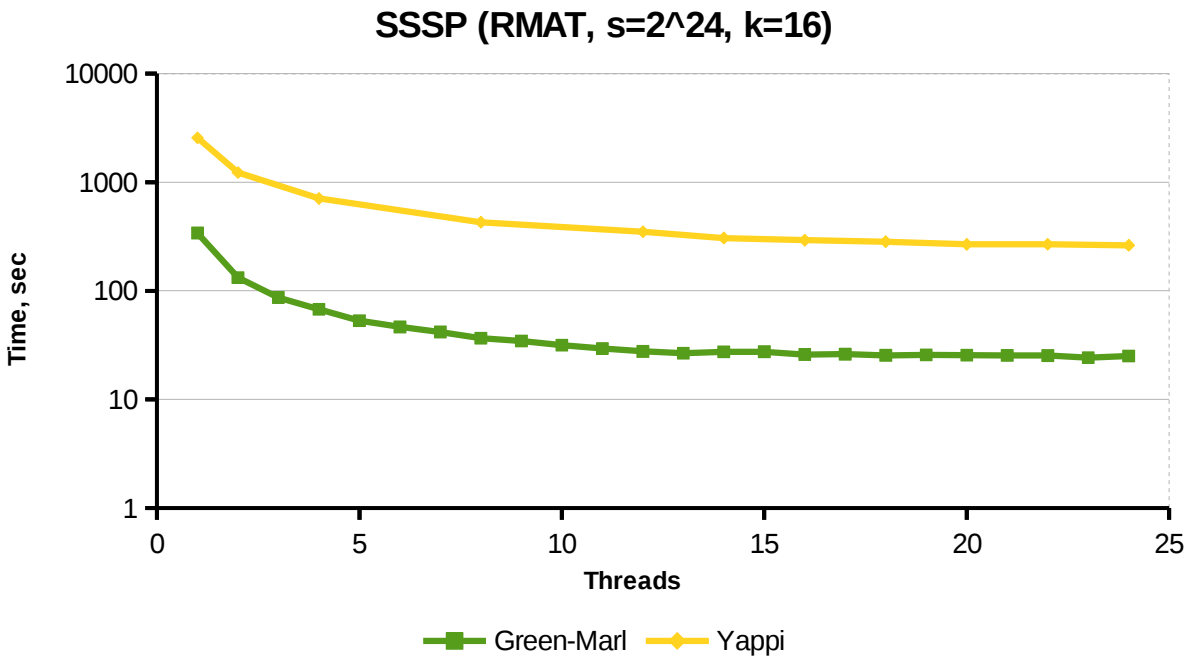
Язык параллельной обработки графов Green-Marl [8]

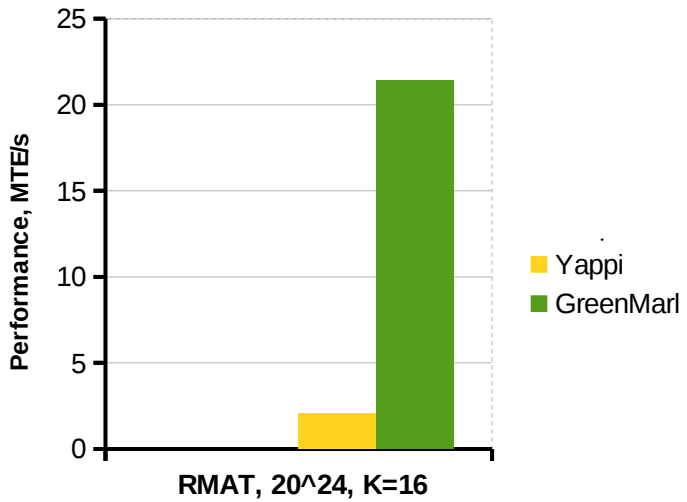
- Разрабатывается в Stanford с 2012 года
- Проблемно-ориентированный язык прогаммирования (DSL)
- Поддержка типов для работы с графами: Graph, Node, Edge и др.
- Параллелизм над данными: линейные итерации (foreach), обходы (InBFS,InDFS)
- Варианты бэкендов: C++/OpenMP, Giraph, GPS, CUDA (в планах)

[8] Sungpack Hong, Hassan Chafi, Edic Sedlar, and Kunle Olukotun. 2012. Green-Marl: a DSL for easy and efficient graph analysis. SIGARCH Comput. Archit. News 40, 1 (March 2012), 349-362

Результаты предварительного оценочного тестирования

Масштабируемость на многопроцессорном вычислительном узле



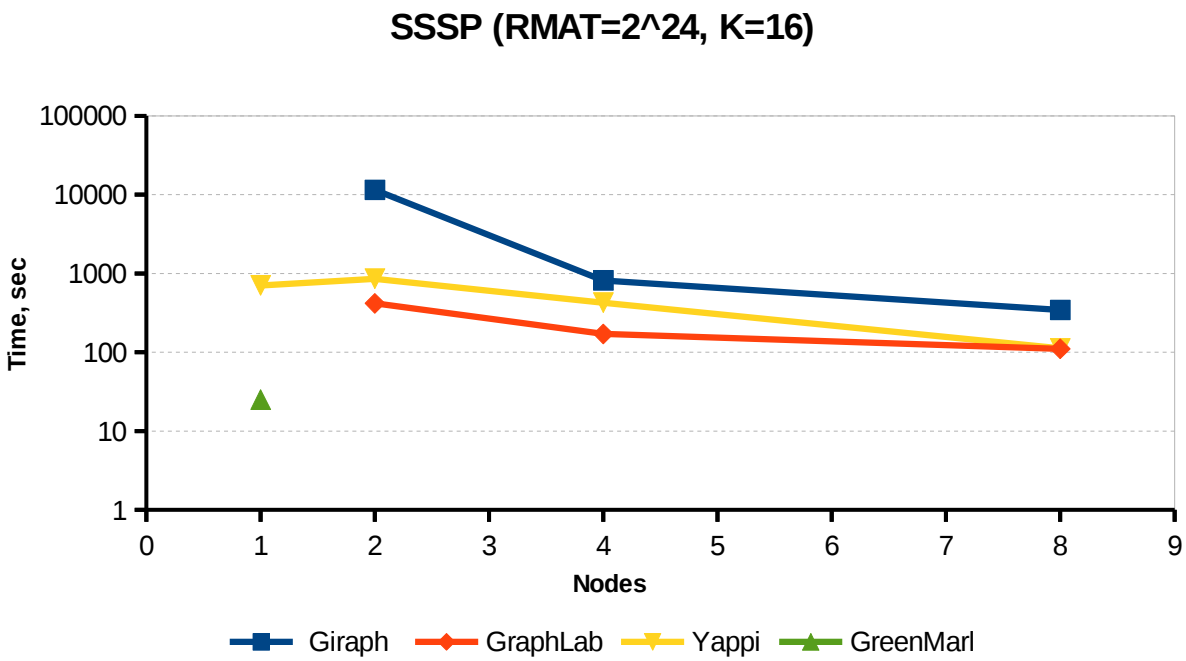


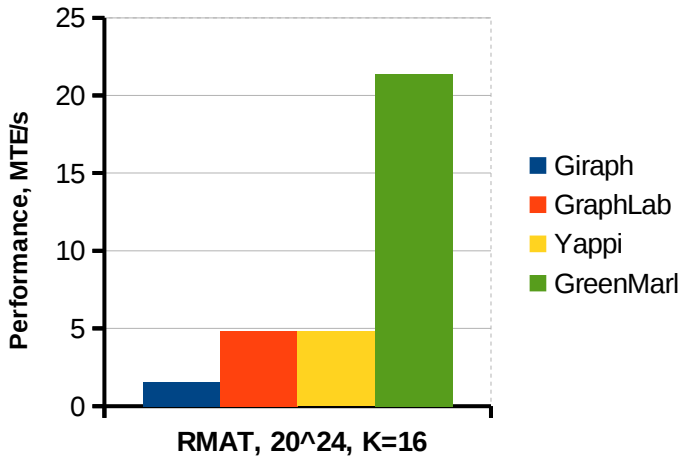
Конфигурация тестовой системы:

- Вычислительный узел – 2×Xeon E5-2630 (12 cores)
- SLES 11.2
- gcc 4.3.4

Результаты предварительного оценочного тестирования

Масштабируемость на кластере





Конфигурация тестовой системы:

- Количество вычислительных узлов – 8
- Вычислительный узел – 2×Xeon E5-2630 (12 cores)
- Интерконнект – dual-port Infiniband FDR, Gigabit Ethernet
- SLES 11.2
- gcc 4.3.4

Выводы

- Размеры реальных графов требуют новых технологий высокоскоростной параллельной обработки графов: как программных так и аппаратных.
- Решение графовых задач на суперкомпьютерах требует специализированных программных моделей (реализованных в виде библиотек, фреймворков, DSL), программирование графовых задач на MPI+OpenMP+CUDA (+SystemVerilog?!) представляется маловозможным.
- Большинство систем на основе модели Pregel ориентированны на распределенные системы (облачные вычисления), а не суперкомпьютеры.
- **Предварительный вывод 1:** Накладные расходы на BSP модель оказали сильное влияние на производительность Giraph, GraphLab, Yappi, в результате чего производительность Green-Marl на одном узле оказалась выше, чем производительность BSP-систем на 8-и узлах.
- **Предварительный вывод 2:** Реализация графового фреймворка на основе модели Pregel требует внесения изменений в концепцию BSP-модели или полный или частичный отказ от BSP в пользу асинхронных моделей (типа GraphLab).

Postscriptum

Ответ одного из разработчиков GraphLab на вопрос об использовании TCP/IP sockets вместо MPI:

Yucheng • Moderator, GraphLab Team • admin

March 4

Hi,

Basically, the MPI API one-sided API is quite difficult to control efficiently for handling very large numbers of small messages. We used to have an MPI communicator based on MPI_Isend before, but it turns out on regular ethernet, a direct TCP socket is faster. Also, we performed most of our testing on EC2, thus Ethernet performance was more important. I do have hopes for a direct ibverbs implementation though.

Yucheng

Спасибо! Вопросы?