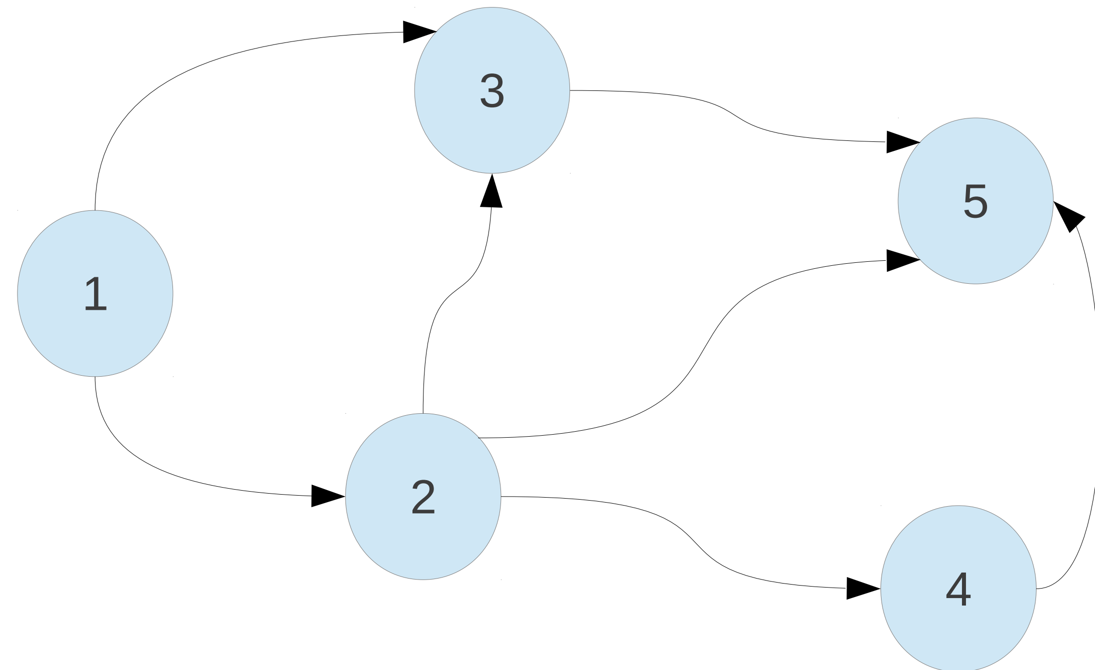
Разработка C++ API для реализации алгоритмов на больших графах

Павел Артёмкин
Руководитель группы

Я. Субботник в Екатеринбурге, 6 июля 2013

- Сколько весит Web-граф
- Модель вычислений
- Алгоритмы на C++

Алгоритм Дейкстры




```
std::queue<vertex*> q;

q.push(get_first_vertex());

while (!q.empty()) {
  vertex* v = q.front();
  q.pop();

  for (auto& e : v->edges) {
    if (e.dst->value == 0 || v->value + e.weight < e.dst->value)
    {
      e.dst->value = v->value + e.weight;
      q.push(e.dst);
    }
  }
}
```

- Применим, если граф можно загрузить в память одного сервера.

Исходные данные

- Ссылочная база — 65 Тб.

Параметры графа:

- 256 млрд. вершин.
- 2 000 млрд. (2 трлн.) рёбер.

Сервера:

- Память 64 Гб, диски —6.2Тб.
- Минимум 10 серверов для хранения базы.
- Имеется 50 серверов для расчётов.

И это только начало...

```
struct addrinfo* result;
struct addrinfo* rp;
int s;

ret = getaddrinfo(NULL, str, &hints, &result);

for (rp = result; rp != NULL; rp = rp->ai_next) {
  s = socket(rp->ai_family, rp->ai_socktype, rp->ai_protocol);

  if (s == -1)
    continue;

  if (bind(s, rp->ai_addr, rp->ai_addrlen) == 0)
    break;

  close(s);
}

if (rp == NULL) {
  fprintf(stderr, "Could not bind\n");
  exit(EXIT_FAILURE);
}

freeaddrinfo(result);
```

Сложности

Передача данных по сети (ненадёжно)

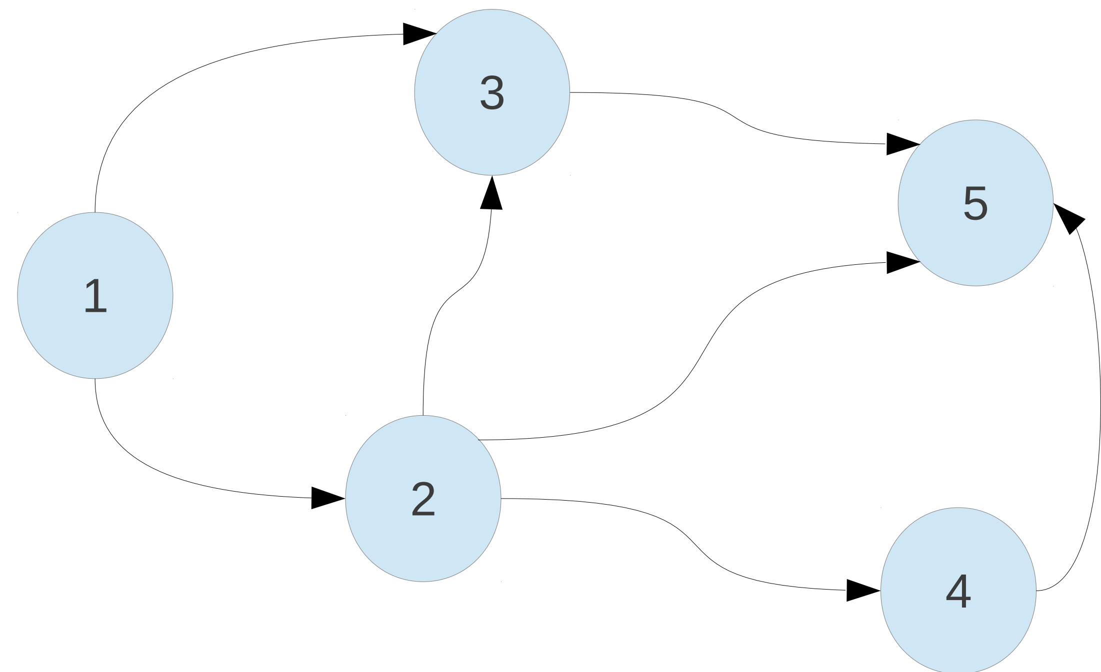
- потеря связности
- перегрузка коммутаторов

Сервера

- выход из строя дисков (ошибки в данных)
- полный отказ сервера (недоступен)

МОДЕЛЬ ВЫЧИСЛЕНИЙ

Алгоритм Дейкстры



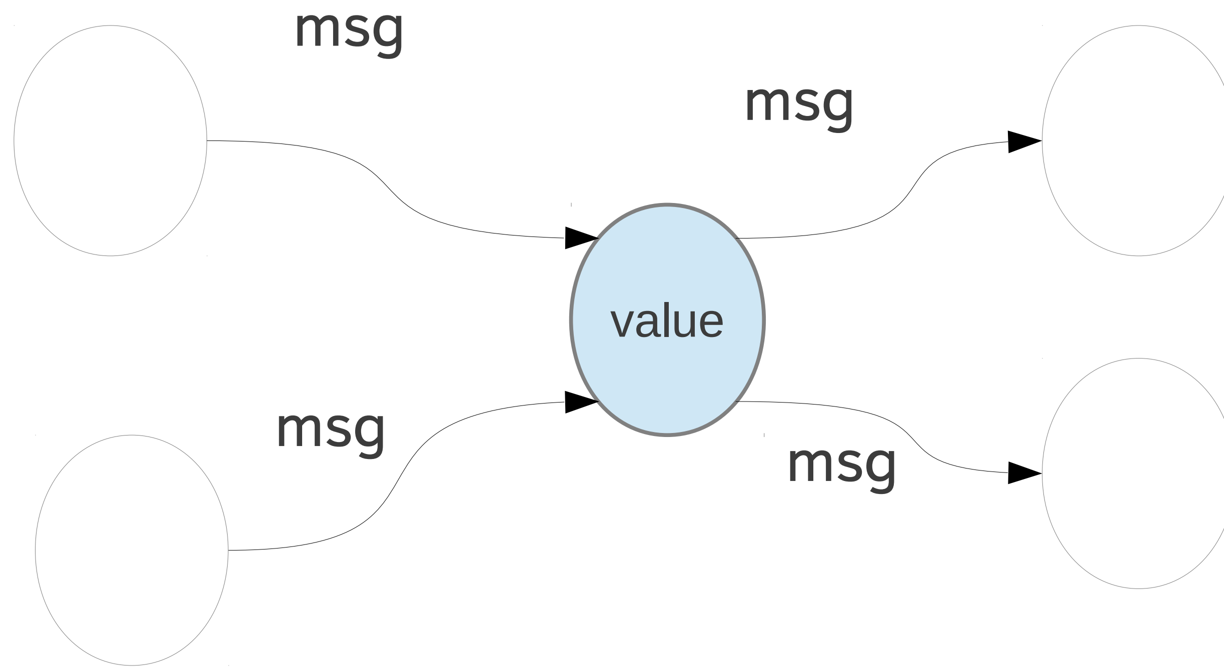

```
std::queue<vertex*> q;

q.push(get_first_vertex());

while (!q.empty()) {
  vertex* v = q.front();
  q.pop();

  for (auto& e : v->edges) {
    if (e.dst->value == 0 || v->value + e.weight < e.dst->value)
    {
      e.dst->value = v->value + e.weight;
      q.push(e.dst);
    }
  }
}
```

- На каждом сервере хранится только часть графа (вершины, исходящие ссылки).



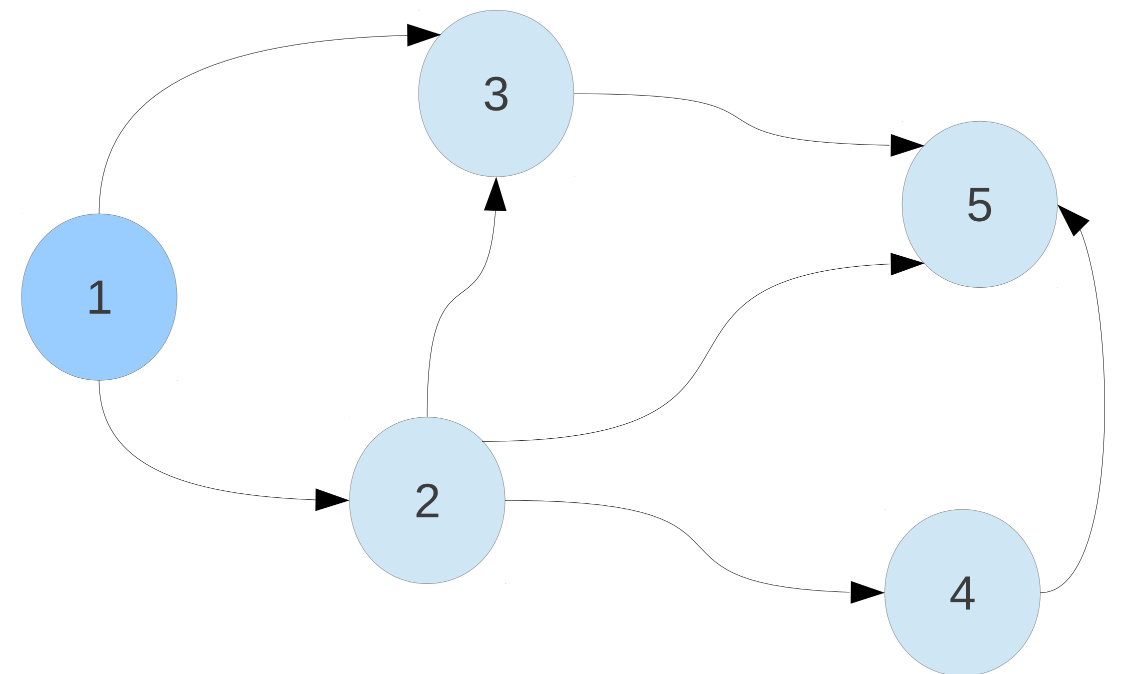

```
void compute (v, msgs) {
  for (auto& m : msgs)
    v->value = min(v->value, m->value);
  for (auto& e : v->edges)
    send(e.key, v->value + e.value);
}
```

Функторы

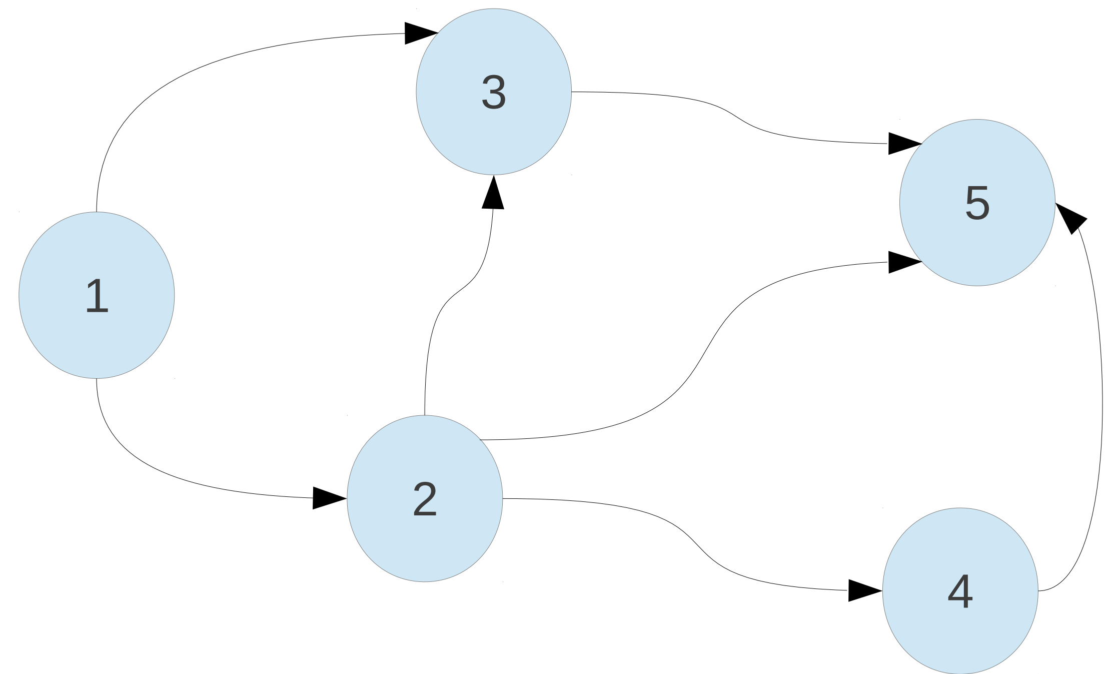
- Load —загрузка данных.
- Compute —вычисления.
- Dump —выгрузка.
- Partition —диспетчеризация сообщений.

Состояние вершины (условие вызова Compute)

- ACTIVE —всегда.
- WAIT —если есть входящие сообщения.
- HALT —никогда.




```
void compute (v, msgs) {
  for (auto& m : msgs)
    v->value = min(v->value, m->value);
  for (auto& e : v->edges)
    send(e.key, v->value + e.value);
}
```



Сообщения

- Передаются между вершинами.
- Можно послать в любую вершину, но в большинстве алгоритмов, между вершинами, соединёнными рёбрами.

C++ API

Обрабатываемые данные

Значения — ключ, вершина, ребро, сообщение.
В разных задачах разные типы (целое, дробное, структура)

Возможные варианты:

- Строки:
  - время на преобразование
  - большой объём
- Динамические типы:
  - сопоставление с типами языка C++
  - собственный язык описание алгоритмов
- Шаблоны C++

Шаблоны C++

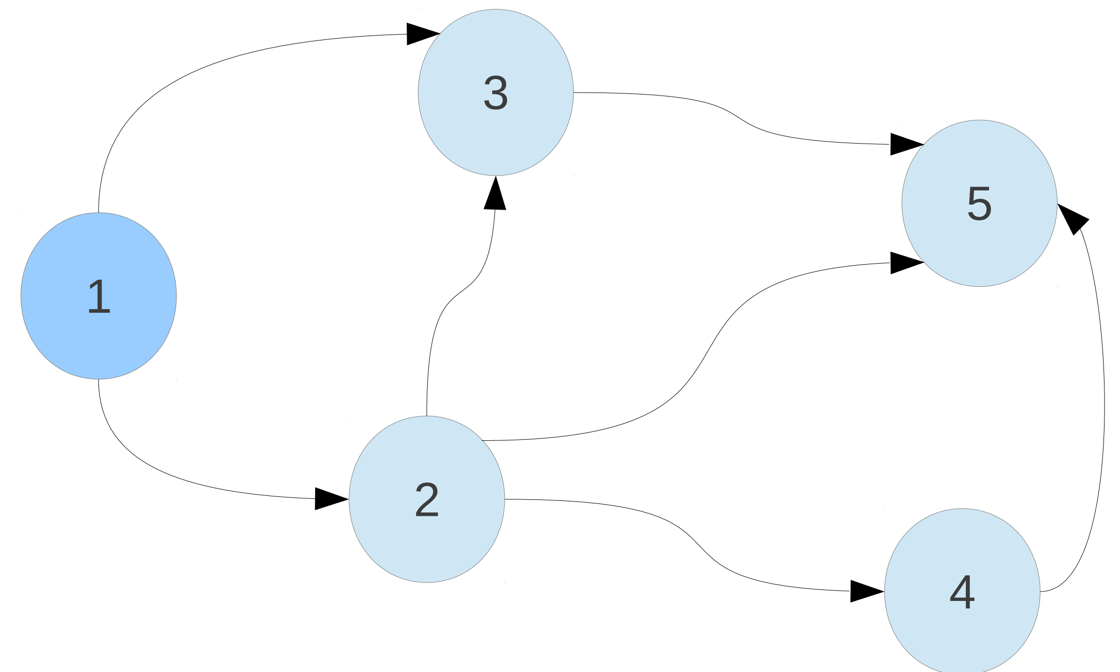
- Получать алгоритмы, оптимизированные под конкретные типы данных.
- Метапрограммирование с использованием шаблонов, позволяет выбрать оптимальный способ сохранения данных, в зависимости от типа:
  - Если типы данных POD (plain old data), то их можно сохранять с помощью memcpy.
  - Если не-POD, то использовать serialize.

КОД

```
template <class TDomain>
struct TLoadFunction {
  virtual void Load(IGraph<TDomain>*) = 0;
};

/// Пример формирования графа
virtual void Load(IGraph<TDomain>* w) {
   w->AddVertexWithValue(1, 0, ACTIVE);
   w->AddVertexWithValue(2, 0);
   w->AddVertexWithValue(3, 0);
   w->AddVertexWithValue(4, 0);
   w->AddVertexWithValue(5, 0);

   w->AddEdgeWithValue(1, 2, 3);
   w->AddEdgeWithValue(1, 3, 5);
   w->AddEdgeWithValue(2, 3, 1);
   w->AddEdgeWithValue(2, 4, 2);
   w->AddEdgeWithValue(2, 5, 7);
   w->AddEdgeWithValue(3, 5, 1);
   w->AddEdgeWithValue(4, 5, 8);
}
```



```
template <class TDomain>
struct TDumpFunction {
  virtual void Dump(IGraph<TDomain>*) = 0;
};

/// Пример вывода результатов
virtual void Dump(IGraph<TDomain>* g) {
  for (auto v = g->GetVertexIterator(); v.Valid(); v.Next()) {
    std::cout << v.Key() << " " << v.Value() << std::endl;
  }
}
```

```
template <class TDomain>
struct TComputeFunction {
    virtual void Compute(TVertexState*, TMessageIterator*,
                         TMessageOutput*) = 0;
};
```

```
/// Интерфейс предоставления доступа к состоянию текущей
/// обрабатываемой вершине в функции Compute
struct TVertexState {
  TV* MutableValue();

  const TKey& Key() const;
  const TV& Value() const;

  TEdgeIterator* GetOutEdgeIterator() const;

  void Activate();
  void Halt();
  void Wait();
};


/// Интерфейс для отправки сообщений и изменения графа
struct TMessageOutput {
    /// Отправить сообщение всем вершинам
    /// из указанной последовательности
    void SendMessageTo(TEdgeIterator* edges, const TM& msg);

    /// Отправить сообщение указанной вершине
    void SendMessageTo(const TKey& key, const TM& msg);
};
```

Дейкстра

```
struct TCompute : TComputeFunction<TDomain> {
  virtual void Compute(TVertexState* v, TMessageIterator* msgs,
                       TMessageOutput* output)
  {
    for (auto& m : msgs)
      *v->MutableValue() = min(v->Value(), m->Value());
    for (auto& e : v->GetOutEdgeIterator())
      output->SendMessageTo(e.Key(), v->Value() + e.Value());
    v->Wait();
  }
};
```

PageRank

```
struct TCompute : TComputeFunction<TDomain> {
  virtual void Compute(TVertexState* v, TMessageIterator* msgs,
                       TMessageOutput* out)
  {
    if (Context().GetTick() >= 1) {
      *v->MutableValue() = 0.15 + 0.85 * Sum(msgs));
    }

    auto edges = v->GetOutEdgeIterator();

    if (edges->Valid())
      out->SendMessageTo(edges, v->Value() / edges->Size());
    else
      out->SendMessageBroadcast(
        v->Value() / Context().GraphInfo().GetVertices()
      );

    if (Context().GetTick() >= Data()->MaxIterations)
      v->Halt();
  }
};
```

Режимы работы (память)

- Все данные графа хранятся в памяти.
- Быстро, но объём обрабатываемых данных, ограничен размером оперативной памяти сервера.

```
template <class TDomain, class TFunctors>
struct TMemoryInstance : IInstance {
  virtual void Load() {
    typename TFunctors::TLoad load;

    load.Load(&MemoryGraph_);
  }
  virtual bool Compute();
  virtual void Dump();
}
```

Режимы работы (диск)

- Данные графа и сообщений хранятся на диске при обработки используются алгоритмы внешней сортировки.
- Медленнее, но позволяет обрабатывать в 50 — 100 раз больший объём данных на том же сервере.

Выполнение

```
template <
  class TDomain,
  class TFunctors,
  class <class D, class F> class TInstance
>
struct TLocalRunner {
  typedef TInstance<TDomain, TFunctors> TInstance;

  void Run(const TInstanceConfig& config) {
    Instance_->Configure(config);
    Instance_->Load();
    while (true) {
      if (!Instance_->Compute())
        break;
    }
    Instance_->Dump();
  }
}
```

Яндекс

Артёмкин Павел

Руководитель группы

stanly@yandex-team.ru

Спасибо

Домен данных

```
template <class K, class V, class E, class M>
struct TLoadFunction {
  virtual void Load(IGraph<K, V, E, M>*) = 0;
}


template <class K, class V, class E, class M>
struct TDomain {
  typedef K TK;
  typedef V TV;
  typedef E TE;
  typedef M TM;
}


typedef TDomain<int, int, int, int> TMyDomain;

struct TLoad : TLoadFunction<TMyDomain> {
  ...
};
```