САМАРСКИЙ УНИВЕРСИТЕТ
SAMARA UNIVERSITY

Построение алгоритмического скелетона для блочной обработки данных в корпоративных гридах из настольных компьютеров

Building an Algorithmic Skeleton for Block Data Processing on Enterprise Desktop Grids

Востокин Сергей Владимирович
Бобылева Ирина Владимировна

Суперкомпьютерные дни в России МОСКВА, 23-24 сентября 2019 г.

Мотивация исследований

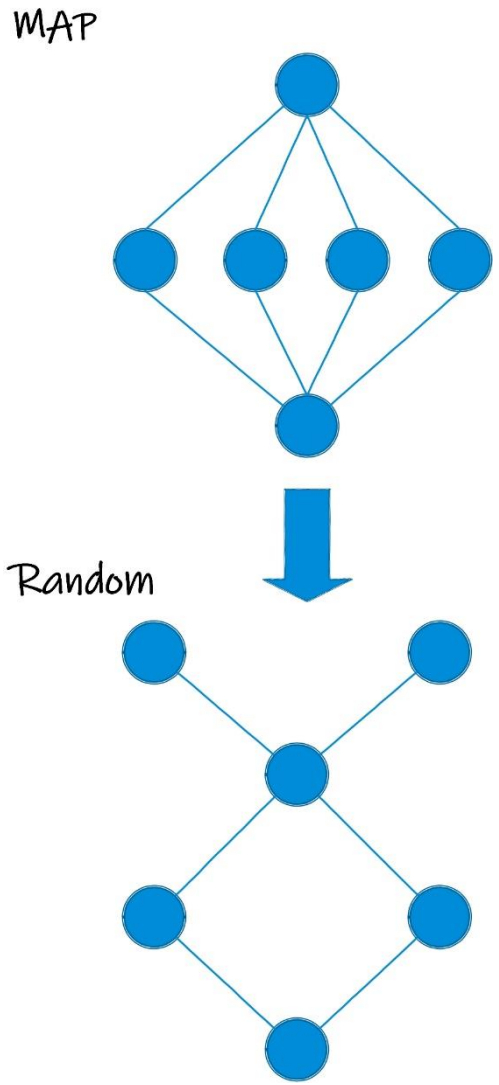
Идеальное распределенное приложение

- способно работать на произвольной группе устройств в сети Интернет:
  - рабочих станциях, ПК, ноутбуках, планшетах, смартфонах, виртуальных машинах в облаке, узлах кластеров и суперкомпьютеров и т.д.;
- обеспечивает рациональное использование вычислительного парка организации;
- имеет неограниченную масштабируемость.

Проблема

Грид системы из настольных компьютеров (десктоп гриды) близки к идеалу распределенного приложения,

но обычно на них решают чрезвычайно параллельные (embarrassingly parallel) задачи.



Можно ли в рамках технических требований десктоп гридов просто и эффективно реализовывать более сложные задачи?

Можно ли снять ограничения на структуру графа зависимостей задач?

Цель и задачи исследования

Цель исследования

- Разработка метода синтеза приложений с произвольным графом зависимостей задач для обработки данных в десктоп гридах.

Задачи исследования

- Описать базовую модель параллельных вычислений для десктоп гридов в виде алгоритмического скелетона.
- На основе модели п.1 описать работу алгоритмического скелетона «портфель задач».
- Путем дальнейшей специализации алгоритмического скелетона «портфель задач» построить новый алгоритмический скелетон «асинхронный круговой турнир» для попарной обработки блочных данных.
- С использованием алгоритмического скелетона «асинхронный круговой турнир» реализовать алгоритм блочной сортировки и исследовать его эффективность.

Метод исследования

Простое описание параллельных вычислений – описание на основе последовательного алгоритма

Для это в работе применяются:

- алгоритмические скелетоны (Murray Cole) – для разделения «параллельной» и «алгоритмической» составляющих вычислительного процесса;
- модель акторов (Carl Hewitt) – для описания «параллельной» составляющей вычислительного процесса в алгоритмическом скелетоне.

Эффективность полученного решения исследуется:

- на программном эмуляторе;
- на базе кластера (MPI + General Parallel File System);
- в десктоп гриде на платформе Everest (разработчик ИППИ РАН).

Алгоритмический скелетон

Алгоритмический скелетон концептуально похож на

обобщенный алгоритм:

```
template <class EuclideanRing>
EuclideanRing gcd(EuclideanRing m, EuclideanRing n){
    while(n!=0){ EuclideanRing t = m % n; m = n; n = t;}
    return m;
}
```

функцию высшего порядка:

```
int f(int x){ return x + 3; }
int g(int (*func)(int), int x){ return func(x) * func(x); }
printf("%d", g(f, 7));
```

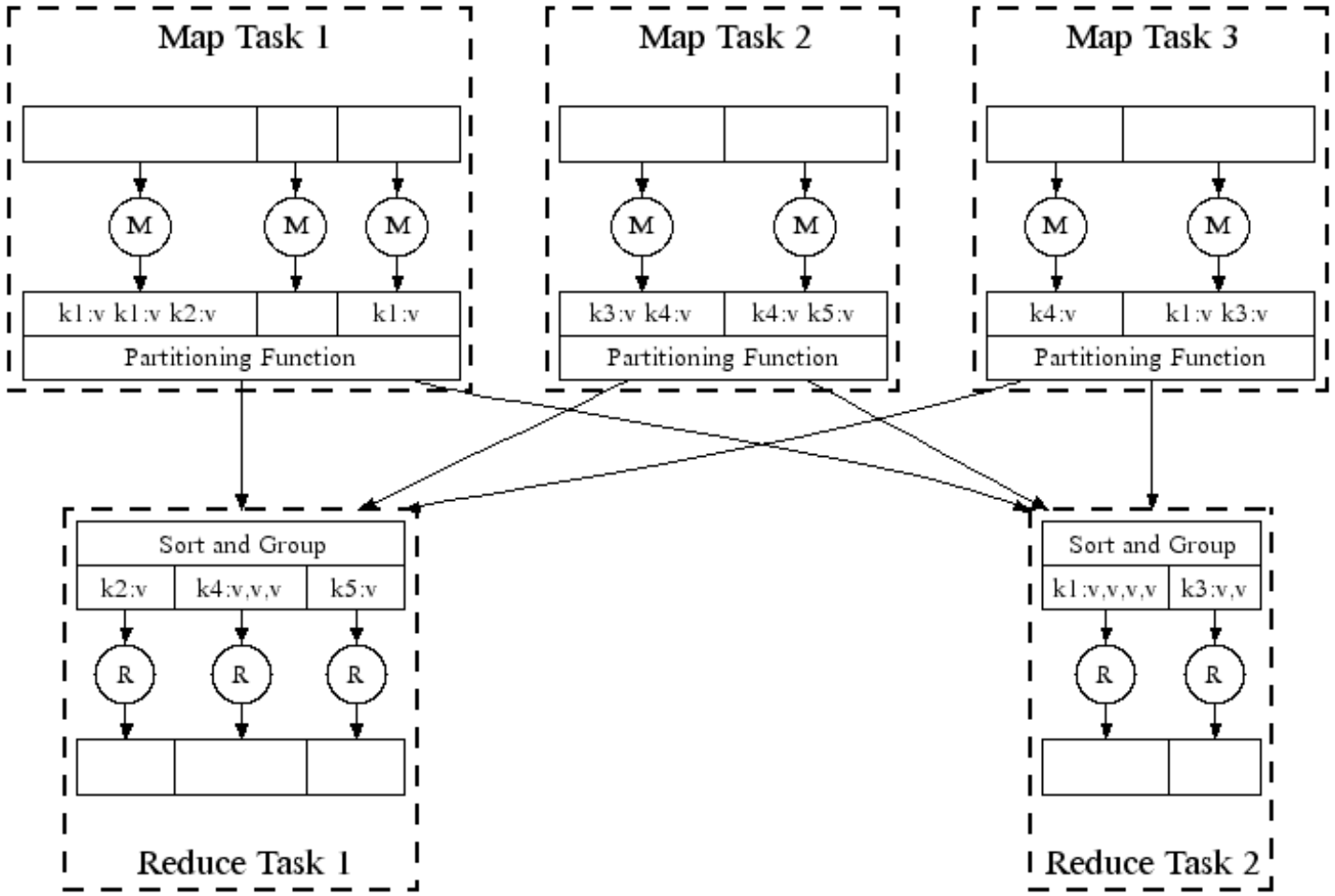
Алгоритмический скелетон (продолжение)

… но реализует не последовательный, а параллельный вычислительный процесс, который описывается некоторой (не алгоритмической) моделью вычислительного процесса.

Например:

```
MapReduce <
        /* параметры – функции */
       map: (in_key, in_value) -> list(out_key, intermediate_value),
       reduce: (out_key, list(intermediate_value)) -> list(out_value)
  > (обычные параметры){ модель вычислений }
```

Модель работы MapReduce



Из Jeff Dean, Sanjay Ghemawat MapReduce: simplified data processing on large clusters (2004)

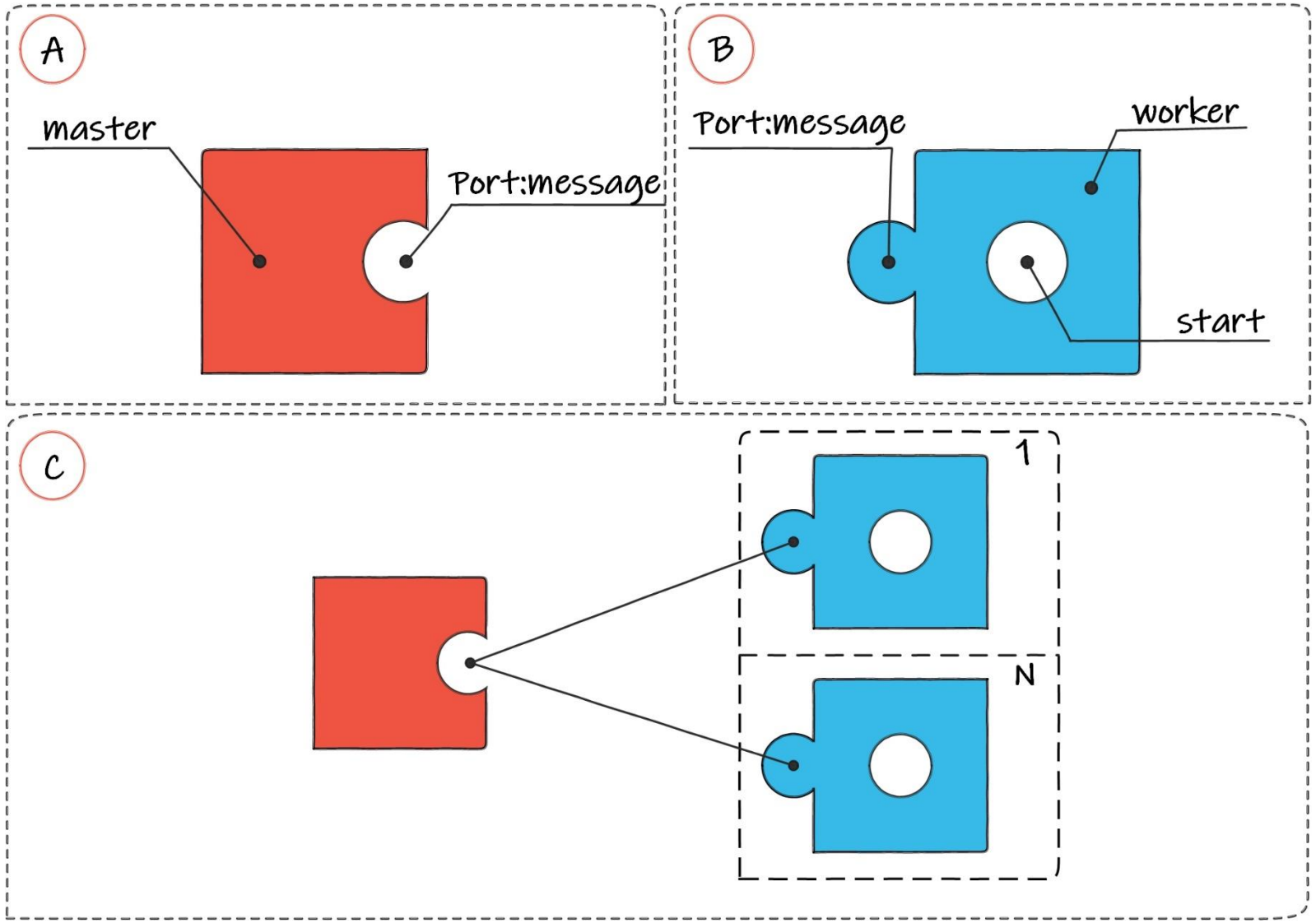
Базовая модель вычислений в десктоп гриде. Скелетон EDG.

```
EDG<
        struct master{void port_handler(message&){ * },
            bool access(message&); * },
        struct worker{ void port_handler(message&m){ * };
            message port; void start(){ * }; * },
        struct message{ send(); * }
>()
```

* – части кода, добавляемые при специализации скелетона

В master::port_handler можно вызывать access(.) и m.send() для сообщений, если access(m) → true; в worker::start и worker:: port_handler можно вызывать port.send().

Модель работы скелетона EDG



Специализация базовой модели: скелетон Bag-of-Tasks (BOT)

BOT<

struct bag{*}; – состояние вычислений (портфель задач)

struct task{*} – состояние задачи (входные данные и результат их обработки)

bool test(bag&){*}; – проверка наличия задачи в текущем состоянии портфеля задач

void get(task&,bag&){*}; – формирование задачи для обработки

void proc(task&){*}; – обработка задачи

void put(task&,bag&){*} – помещение результата обработки задачи в портфель

>()

Описание работы скелетона Bag-of-Tasks (BOT)

Семантика скелетонов, производных от скелетона EDG, описывается только в терминах алгоритмов и структур данных:

```
struct message{task t; bool is_first; }


struct worker{
        void start(){port.is_first=true; port.send();}
        void port_handler(message&m){proc(m.t); m.send();}.
}
```

Описание работы скелетона Bag-of-Tasks (BOT) (продолжение)

```
struct master{
        bag b; list<message*> wait;
        void port_handler(message&m){ Step_1; Step_2; Step_3; }.
}

Step_1: if(m.is_first) m.is_first=false; else put(m.t,b); wait.push_back(&m).

Step_2: while(!wait.empty() && test(b)){
                message*m=wait.back(); wait.pop_back();
                get(m->t, b); m->send();
        }.

Step_3: if(wait.size()==N) stop().
```

Скелетон «асинхронный круговой турнир» (ART)

ART<void prepare(int team); void play(int team_i, int team_j)>(int M, int I1[M][M], int I2[M][M])

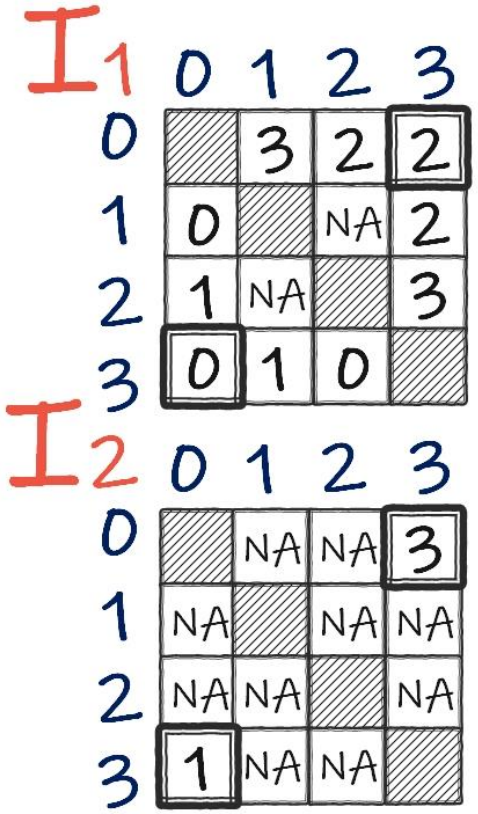
Скелетон ART реализует параллельный эквивалент последовательного алгоритма спортивного кругового турнира.

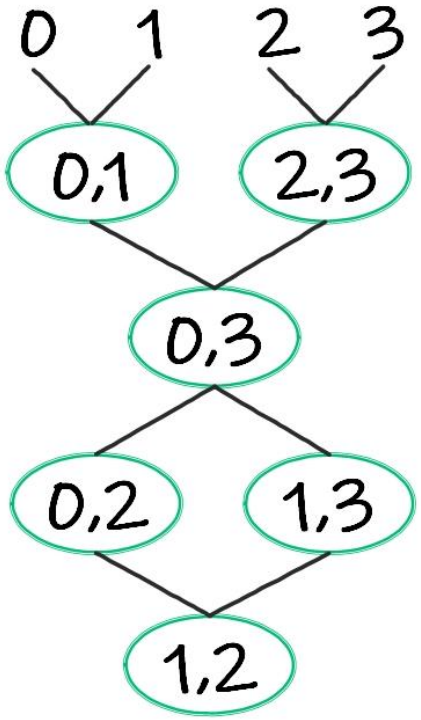
Например:

```
for (int i = 0; i < M; i++) prepare(i);
for (int i = 1; i < M; i++) for (int j = 0; j < i; j++) play(j, i);
```

Порядок операций play(.) может быть произвольным и зависит от цели использования скелетона (например, сортировка).

Кодирование графа зависимостей между операциями play(.).
Пример турнира четырёх команд.





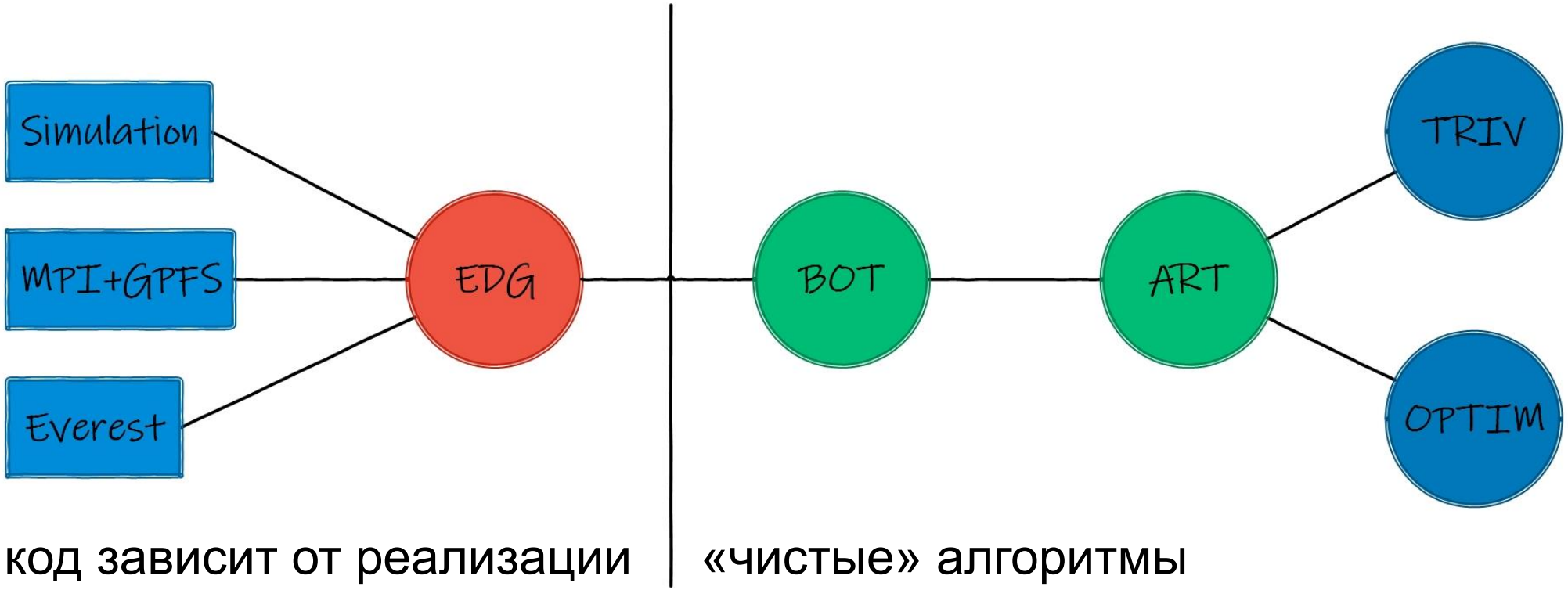
Описание работы скелетона «асинхронный круговой турнир» (ART) (продолжение)

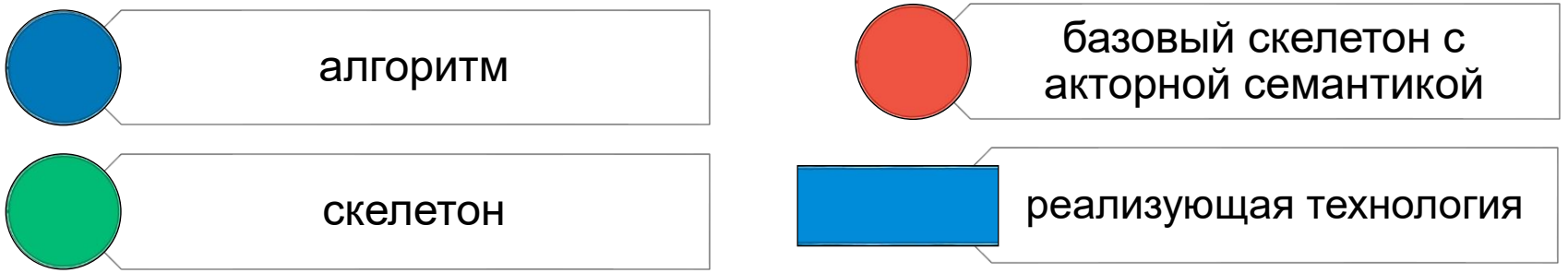
```
struct bag{
        int D[M][M];
        list<pair<int,int>> games;
        int I1[M][M]; int I2[M][M]; int S[M][2];
}
struct task{int i; int j;}

bool test(bag&b){return !b.games.empty();}

void get(task&t,bag&b){
        pair<int,int>p=b.games.back(); b.games.pop_back();
         t.i=p.first;t.j=p.second;
}
void proc(task&t){if(t.j==NA) prepare(i); else play(t.i,t.j);}
```

Описание работы скелетона «асинхронный круговой турнир» (ART) (продолжение)

```
void put(task&t,bag&b){
    if (t.j==NA){// prepare task
        if (--D[S[t.i][0]][S[t.i][1]] == 0) {
            std::pair<int, int> p(S[t.i][0], S[t.i][1]);
                b.games.push_back(p);
                }
    else{// play task
        if (I1[t.i][t.j] != NA) {
            if (--D[I1[t.j][t.i]][I1[t.i][t.j]] == 0) {
                std::pair<int, int>
                    p(I1[t.j][t.i], I1[t.i][t.j]);
                        b.games.push_back(p);
                }
            }
        if (I2[t.i][t.j] != NA) {
            if (--D[I2[t.j][t.i]][I2[t.i][t.j]] == 0) {
                std::pair<int, int>
                    p(I2[t.j][t.i], I2[t.i][t.j]);
                        b.games.push_back(p);
                }
            }
        }
    }
```

Общая схема методики синтеза алгоритмов





Тестовая задача – блочная сортировка. Алгоритмы TRIV и OPTIM.

Вход: массив целых чисел, разбитый на блоки размером ~ 720MB.

Выход: отсортированный массив целых чисел, разбитый на блоки размером ~ 720MB.

Определение prepare(i) в скелетоне ART:

сортировка i-ого блока в памяти алгоритмом std::sort (C++).

Определение play(i,j) в скелетоне ART:

предусловие: блоки i и j – отсортированы;

слияние блоков i и j в промежуточный блок b в памяти алгоритмом std::merge (C++);

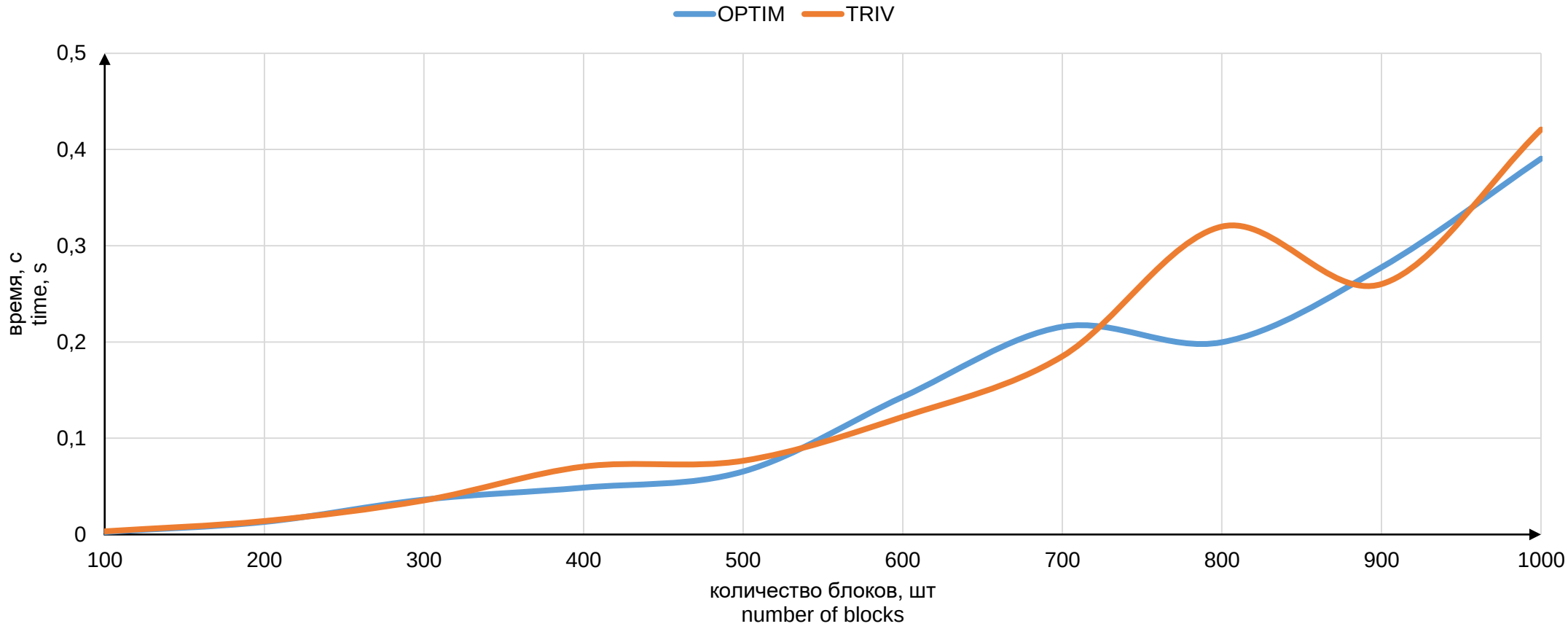
разбиение блока b пополам, запись младшей части в i-ый блок, а старшей – в j-ый блок.

Матрицы I1 и I2 определяют алгоритм TRIV или OPTIM,

блоки хранятся в отдельных файлах.

Результаты экспериментов

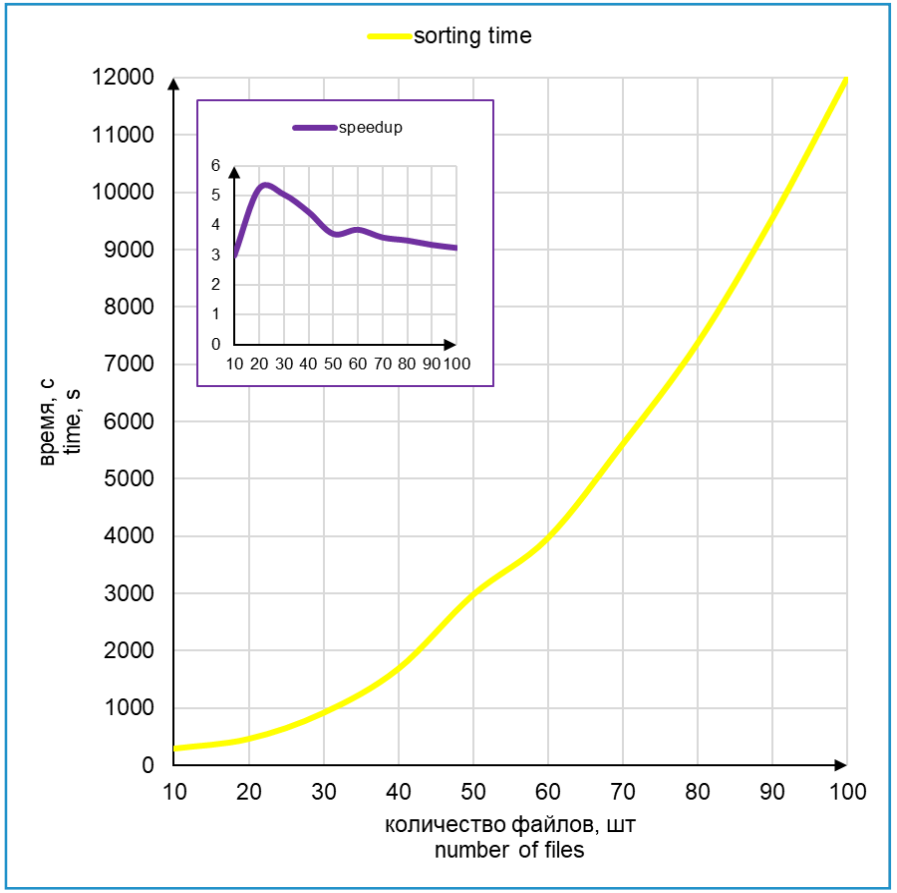
Зависимость времени выдачи всех задач турнира от количества блоков в наборе данных



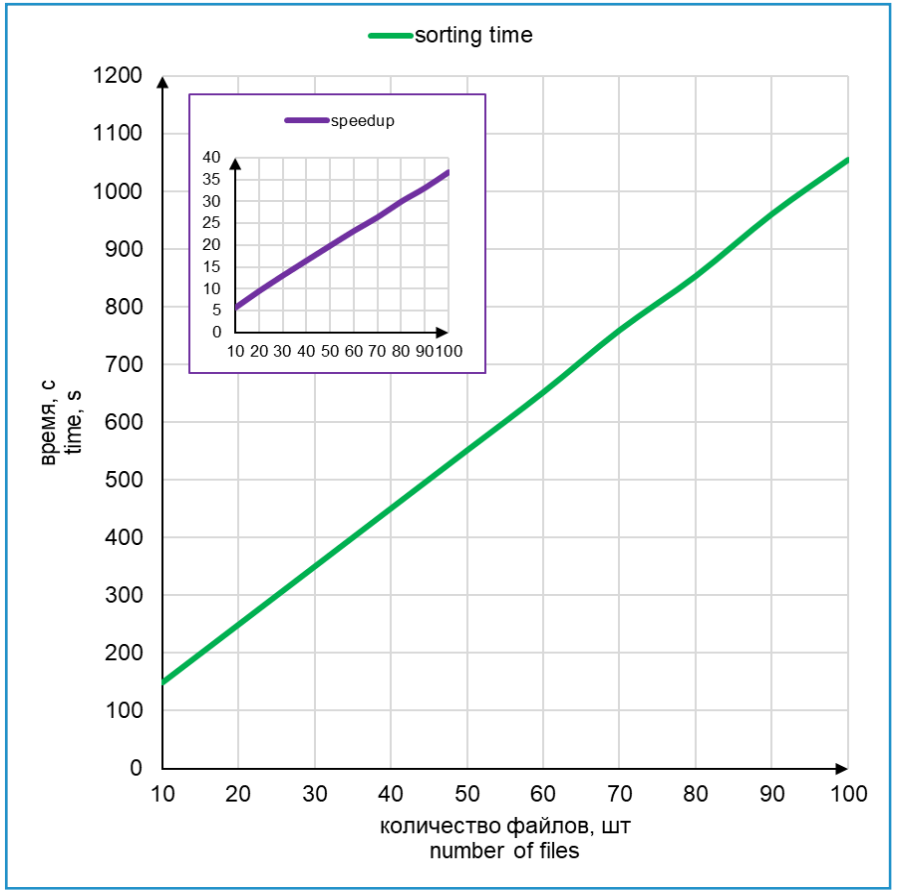
Конфигурация: Intel(R) Core(TM) i3 - 3220T CPU @ 2.80GHz PC with 4 GB of RAM, 4 worker processes

Результаты экспериментов (продолжение)

Зависимость времени сортировки от числа файлов



Зависимость времени сортировки от числа файлов (имитационный эксперимент)



Конфигурация узлов: 2x Intel Xeon X5560, 2.80GHz, 8MB Cache, 12MB RAM, torque, MPI, GPFS

Тестирование блочной сортировки в десктоп гриде на платформе Everest

Конфигурация

- Приложение сортировки (оркестратор): ноутбук с подключением к сети Интернет.
- Управляющий сервер: https://everest.distcomp.org.
- Рабочие узлы (всего 10): агенты Everest на виртуальных машинах в облаке Самарского университета, развертывание из браузера по адресу https://vss.ssau.ru/.
- Массив данных: файлы по ~300MB на разделяемом диске \\MSCS-FS\data\studview\everest.

Предварительные результаты

- Сортировка 10 файлов, локальный диск, 1 узел, с учетом предварительного копирования всех файлов на узел (~ 41 секунда) – 804,16 секунд.
- Сортировка 10 файлов, 1 узел, непосредственно на разделяемом диске – 936.27 секунд.
- Сортировка 10 файлов на 10 узлах на разделяемом диске – 589.45 секунд.
- Ускорение 1.59 относительно работы с разделяемым диском на 1 узле.
- Ускорение 1.36 относительно работы с локальным диском на 1 узле.

Основные результаты

Предложен метод на основе алгоритмических скелетонов, который можно использовать для разработки десктоп грид приложений со сложными зависимостями между задачами.

Экспериментально подтверждена возможность эффективного исполнения приложений, разработанных на базе алгоритмических скелетонов, при решении задач с интенсивным обменом данными в десктоп гриде.

Преимущества метода:

- абстрагирование от реализации десктоп грида — подходит для различных платформ;
- алгоритмическое описание — обеспечивает удобство понимания с одной стороны, независимость от языка программирования - с другой;
- иерархия скелетонов, построенная по принципу специализации, — обеспечивает высокое повторное использование кода и снижает сложность программирования прикладных алгоритмов.

Показано, что

- передача ссылок на объекты данных в виде имен файлов в распределенной файловой системе (вместо непосредственной передачи данных между управляющим и рабочими процессами) обеспечивает ускорение вычислений.

Вопросы дальнейших исследований

1. Каким должен быть базовый скелетон и его математическое описание?

Желательно иметь более строгое математическое описание работы базового скелетона.

2. Как работать с глобальными распределенными файловыми системами, например, IPFS (Juan Benet)?

Интересны технические аспекты реализации взаимодействия с IPFS и математические аспекты, связанные с балансировкой нагрузки в приложениях.

САМАРСКИЙ УНИВЕРСИТЕТ
SAMARA UNIVERSITY

БЛАГОДАРЮ ЗА ВНИМАНИЕ