



**SAMARA** UNIVERSITY

Sergey Vostokin

Samara National Research University  
named after academician S.P. Korolev

**USING DISTRIBUTED SHARED MEMORY  
BASED ON WRITE-AHEAD LOGGING FOR  
COMPUTING ON DESKTOP GRIDS**

**International Conference “Russian Supercomputing Days”**

Moscow, Shuvalovsky building of the Moscow State University,  
September 28-29, 2026



**SAMARA** UNIVERSITY

Востокин С.В.

Самарский национальный исследовательский университет  
имени академика С.П. Королева

**ИСПОЛЬЗОВАНИЕ РАСПРЕДЕЛЕННОЙ  
РАЗДЕЛЯЕМОЙ ПАМЯТИ НА ОСНОВЕ УПРЕЖДАЮЩЕЙ  
ЖУРНАЛИЗАЦИИ ДЛЯ ВЫЧИСЛЕНИЙ В ГРИД-  
СИСТЕМАХ ИЗ НАСТОЛЬНЫХ КОМПЬЮТЕРОВ**

Международная конференция «Суперкомпьютерные дни в России»  
г. Москва, Шуваловский корпус МГУ, 28 – 29 сентября 2026 года



1. Мотивация исследования.
2. Реализация модели распределенной общей памяти на основе механизма упреждающей журнализации в виде прикладной библиотеки Templet.
3. Методика сравнительного тестирования разрабатываемой библиотеки и современных решений для грид-систем из настольных компьютеров.
4. Результаты сравнительного тестирования пропускной способности разрабатываемой библиотеки Templet и платформы Everest ИППИ РАН.
5. Обсуждение результатов тестирования.
6. Заключение.



Востребованность использования временно простаивающей инфраструктуры из настольных компьютеров для выполнения ресурсоемких расчетов подтверждается многолетним опытом успешной эксплуатации систем **BOINC** и **HTCondor**.

**Проблема** – *ограничения*, связанные с особенностями дизайна данных систем под требования компьютерного оборудования 90-х - 2000-х годов.

Основа дизайна – архитектура многозадачных вычислений, которая обеспечивает отказоустойчивость, масштабируемость, миграцию нагрузки.

**Однако**

- ☐ архитектура на основе задач является *недостаточно производительной*,
- ☐ а также *недостаточно гибкой и удобной для программирования*.



Альтернатива – **паттерн упреждающей журнализации** – **write-ahead logging (WAL)**.

**Принцип WAL:** изменения состояния вначале записываются в журнал, а затем последовательно считываются из него и применяются к текущему состоянию для получения следующего.

**Применение WAL:** внутренний механизм СУБД, брокер сообщений Kafka и др.

**WAL** реализует **распределенную разделяемую память** – **distributed shared memory (DSM)**.

**DSM** – удобна для решения прикладных задач, так как

- она соответствует модели выполнения современных промышленных языков программирования;
- максимально скрывает от программиста технические проблемы, связанные с вычислениями в неспециализированных распределенных средах.

### Что требуется для применения DSM на базе WAL?

- ❑ Конкретная реализация модели DSM на основе интерфейса записи/чтения событий WAL.
- ❑ Методика для сравнительного тестирования DSM на основе WAL и известных решений.



```
1 class base_bag_of_tasks{
2 public:
3     void resize(unsigned size){ <...> }
4     void add(unsigned id,int n){ <...> }
5     bool ready_to_get(){ <...> }
6     bool get(unsigned& id, int& n){ <...> }
7     void put(unsigned id,int nxn){ <...> }
8 public:
9     std::vector<int> N;
10    std::vector<int> NxN;
11 private:
12    std::set<unsigned> unprocessed;
13    bool _ready_to_get;
14    unsigned get_rand_unprocessed(){ <...> }
15 };
```

разделяемое (общее) состояние



```
1   base_bag_of_tasks tbag;
2
3   tbag.resize(SIZE);
4   for(int i=0;i<SIZE;i++) tbag.add(i,i);
5
6   while(!tbag.ready_to_get())/*wait*/;
7
8   unsigned id; int N, NxN;
9   while(tbag.get(id,N)){
10      NxN = N*N; tbag.put(id,NxN);
11  }
12  for(int i=0;i<SIZE;i++)
13      std::cout << tbag.N[i] <<"^2 = "
14              << tbag.NxN[i] << std::endl;
```

разделяемое (общее) состояние

подчеркнут код «задачи»



```
1 void base_bag_of_tasks::add(unsigned id,int n){
2     N[id]=n; unprocessed.insert(id);
3     if(unprocessed.size()==N.size())_ready_to_get=true;
4 } //--- modified to -->
5 void bag_of_tasks::add(unsigned id,int n){
6     update(_add, [&](std::ostream&out) {
7         out << id << " " << n;
8     },
9     [this](std::istream&in) {
10         unsigned id; int n; in >> id >> n;
11         N[id]=n; unprocessed.insert(id);
12         if(unprocessed.size()==N.size())_ready_to_get=true;
13     });
14 }
```

код без адаптации  
к модели DSM

добавленный код,  
реализующий функционал DSM



```
1  bag_of_tasks tbag(wal);
2  if(pid==0 && !tbag.ready_to_get()){// in master 'process'
3      tbag.resize(SIZE);
4      for(int i=0;i<SIZE;i++) tbag.add(i,i);
5  }
6  while(!tbag.ready_to_get())/*wait*/;
7
8  unsigned id; int N, NxN;
9  while(tbag.get(id,N)){
10     NxN = N*N; tbag.put(id,NxN);
11 }
12 if(pid==0){// in master 'process'
13     for(int i=0;i<SIZE;i++) std::cout << tbag.N[i]
14         <<"^2 = " << tbag.NxN[i] << std::endl;
15 }
```

подчеркнут код, добавленный  
в распределенную версию программы



```
1 class mapper{                                «инъекция зависимостей»
2 public:
3     class engine{                            механизм инициализации для конкретной реализации
4         void on_init(unsigned size){_mapper->on_init(size);}
5     protected:
6         mapper* _mapper;                    <...>
7     };
8 public:
9     mapper(mapper::engine& eng):_engine(eng){eng._mapper=this;}
10    void init(unsigned size){_engine.init(size);} <...>
11 protected:
12     virtual void on_init(unsigned size) = 0;    <...>
13 private:
14     mapper::engine& _engine;                  код инициализации конкретного алгоритма
15 };
```

```
1 class basic_engine: public mapper::engine{
2     void init(unsigned size)override{_size=size;on_init(size);}
3     void map()override{
4         for(int id=0;id<_size;id++)
5             {io_test(id,false); on_map(id); io_test(id,true);}
6     }
7     void io_test(unsigned id,bool mapped){
8         std::stringstream sstr;
9         on_save(id,sstr,mapped);on_load(id,sstr,mapped);
10    }
11    unsigned _size;    фрагменты алгоритма, интегрированные в механизм выполнения
12 };
13 class sync_state_engine: public mapper::engine{ <...> };
14 class everest_engine: public mapper::engine{ <...> };
```



```
1 class throughput_test_mapper: public mapper{
2 public:
3     throughput_test_mapper(mapper::engine& eng): mapper(eng){}
4     std::vector<int> N; std::vector<int> NxN;
5 private:
6     void on_init(unsigned size) override{N.resize(size);NxN.resize(size);}
7     void on_map(unsigned id) override{ NxN[id] = N[id] * N[id]; }
8     void on_save(unsigned id, std::ostream& out, bool mapped) override{
9         if(mapped) out << NxN[id]; else out << N[id];
10    }
11    void on_load(unsigned id, std::istream& in, bool mapped) override{
12        if(mapped) in >> NxN[id]; else in >> N[id];
13    }
14 };
```

специализация обобщенного алгоритма «применить ко всем» (map) для вычисления  $n \times n$



```
1  basic_engine eng;           универсальный код для тестирования механизмов выполнения
2  //sync_state_engine eng;
3  //everest_engine eng;
4  throughput_test_mapper a_mapper(eng);
5  if(master){
6      a_mapper.N.resize(SIZE);
7      for(int i=0;i<SIZE;i++) a_mapper.N[i] = i;
8      a_mapper.init(SIZE);
9  }
10 a_mapper.map();
11 if(master){
12     for(int i=0;i<SIZE;i++)
13         std::cout << a_mapper.N[i]
14             << "^2 = " << a_mapper.NxN[i] << std::endl;
15 }
```

**«инъекция зависимостей»**

**Таблица 1.** Обработка 1500 операций  $f(x) = x^2$  в системах Templet и Everest с использованием 500 слотов.

| Режим                        | Опыт 1<br>время, с | Опыт 2<br>время, с | Опыт 3<br>время, с | Среднее<br>время, с | Пропуск-<br>ная способ-<br>ность, $s^{-1}$ | Оборотное<br>время, с |
|------------------------------|--------------------|--------------------|--------------------|---------------------|--------------------------------------------|-----------------------|
| Everest(raw) <sup>1</sup>    | 346,558            | 348,601            | 347,816            | 347,658             | 4,314                                      | 115,886               |
| Everest(parcel) <sup>2</sup> | 92,278             | 97,167             | 91,421             | 93,622              | 16,021                                     | 31,207                |
| Templet(sync) <sup>3</sup>   | 12,783             | 17,124             | 11,284             | 13,731              | 109,241                                    | 4,577                 |
| Templet(lazy) <sup>4</sup>   | 0,322              | 0,658              | 1,119              | 0,700               | 2141,832                                   | 0,233                 |

**Таблица 2.** Обработка 15000 операций  $f(x) = x^2$  в системах Templet и Everest с использованием 500 слотов.

| Режим                         | Опыт 1<br>время, с | Опыт 2<br>время, с | Опыт 3<br>время, с | Среднее<br>время, с | Пропуск-<br>ная способ-<br>ность, $s^{-1}$ | Оборотное<br>время, с |
|-------------------------------|--------------------|--------------------|--------------------|---------------------|--------------------------------------------|-----------------------|
| Everest(parcel) <sup>5</sup>  | 93,253             | 98,394             | 94,316             | 95,321              | 157,362                                    | 3,177                 |
| Templet(sync) <sup>6</sup>    | 186,866            | 236,094            | 260,207            | 227,722             | 65,869                                     | 7,590                 |
| Templet(lazy) <sup>7</sup>    | 32,193             | 176,583            | 74,403             | 94,393              | 158,909                                    | 3,146                 |
| Templet(recover) <sup>8</sup> | 1,327              | 1,776              | 1,050              | 1,384               | 10831,732                                  | 0,046                 |

Условные обозначения экспериментальных режимов выполнения на таблицах 1 и 2.

- 
- <sup>1</sup> Everest(raw): каждая операция  $f(x) = x^2$  выполняется в отдельной задаче, всего 1500 задач.
  - <sup>2</sup> Everest(parcel): 3 операции  $f(x) = x^2$  выполняется в задаче, всего 500 задач.
  - <sup>3</sup> Templet(sync): данные копируются на диск при каждом обращении к журналу для записи.
  - <sup>4</sup> Templet(lazy): копирование данных на диск выполняется в отложенном (lazy) режиме.
  - <sup>5</sup> Everest(parcel): 30 операций  $f(x) = x^2$  выполняется в одной задаче, всего 500 задач.
  - <sup>6</sup> Templet(sync): данные копируются на диск при каждом обращении к журналу для записи.
  - <sup>7</sup> Templet(lazy): копирование данных на диск выполняется в отложенном (lazy) режиме.
  - <sup>8</sup> Templet(recover): время считывания 15000 ранее выполненных задач при восстановлении итогового состояния вычислений.

1. Высокие накладные расходы на управление многозадачными вычислениями требуют объединения задач в пакеты ([таблица 1](#), сравнение Everest(raw) и Everest(parsels)). При этом в случае использования 500 пакетов в 500 слотах рекомендуемое время выполнения одной задачи (для достижения ~50% эффективности) составляет **не менее ~30 секунд**.
2. Реализация на основе WAL способна (в режиме 1500 задач на 500 процессов, [таблица 1](#)) обеспечивать более высокую (на порядок) пропускную способность. При этом рекомендуемое время выполнения одной «задачи» – **не менее ~5 секунд**.
3. При значительном количестве задач ([таблица 2](#), 15000 задач) производительность реализации WAL падает, в то время как производительность Everest можно стабилизировать за счет объединения задач в пакеты. Причина – это большое количество операций записи в журнал.
4. Для 15000 задач WAL может догнать Everest по пропускной способности, если использовать режим асинхронной/отложенной записи ([таблица 2](#), Templet(lazy)).
5. Если в режиме асинхронной/отложенной записи произойдет сбой управляющего узла непосредственно перед завершением расчета ([таблица 2](#), Templet(recover)), восстановление состояния всех процессов из журнала займет ~1,5 секунды.

1. Продемонстрирована реализация высокоуровневой модели программирования с разделяемой памятью (DSM) на базе интерфейса записи/чтения журнала событий (WAL).
2. Предложена методика сравнительного нагрузочного тестирования реализаций DSM на основе WAL и традиционных реализаций на основе многозадачных вычислений.
3. Показано, что по параметру пропускной способности предложенная реализация DSM на основе WAL не уступает многозадачной реализации Everest (ИППИ РАН), оптимизированной для грид-систем из настольных компьютеров организации. Более того, длительность «задач» может иметь порядок секунд, что заметно упрощает управление вычислениями и уменьшает потери времени на восстановление после отказа.

Полученные результаты направлены на расширение областей применения и повышение эффективности использования вычислительной инфраструктуры из неспециализированных компьютеров, в частности, в университетах и научно-исследовательских организациях, где решаются ресурсоемкие вычислительные задачи.

<http://templet.ssau.ru/wiki> - вики и образовательные ресурсы проекта Templet  
<https://github.com/the-templet-project/templet/tree/master/samples/maptest>  
– исследуемый код и результаты экспериментов

**Автор:** *Востокин Сергей Владимирович* ([vostokin.sv@ssau.ru](mailto:vostokin.sv@ssau.ru))  
д.т.н., профессор кафедры программных систем,  
Самарский национальный исследовательский университет  
имени академика С.П. Королева

**СПАСИБО ЗА ВНИМАНИЕ !**