Distributed Tasks: Introducing Distributed Computing to Programming Competitions

Adam Karczmarz, Jakub Łącki, Adam Polak,
Jakub Radoszewski, Jakub O. Wojtaszczyk

University of Warsaw and Google Warsaw, Poland

Olympiads in Informatics *conference, Kazan, Russian Federation*

Background

- Distributed (and cloud) computing is gaining much attention nowadays.
- The computing giants of today (e.g. Google, Amazon, Facebook) do not operate on enormous mainframes, but on networked farms of smaller servers.
- Major programming competitions (e.g. the IOI, the ACM ICPC, Google Code Jam, TopCoder's algorithmic track, Facebook Hacker Cup, the CodeForces) did not include any concepts of distributed computing.
- This work resulted in the introduction of distributed tasks in a major Polish programming competition **Algorithmic Engagements** (*Potyczki Algorytmiczne*) and of a distributed track at **Google Code Jam**.

Distributed Tasks

Model: $M = 100$ nodes (machines) connected by a network.

Input: Every node has random access to input data. Usually the whole data is too large to be accessed on a single node.

Communication: The nodes can exchange messages using a very simple C++ API of *put-send* and *receive-get* methods.

Output: Exactly one node writes the results to the standard output.

C++ API

The same code is ran on all nodes.

```
int NumberOfNodes();
int MyNodeId(); // 0 .. NumberOfNodes() - 1
```

Constructing messages:

```
void PutChar(int target, char value);
void PutInt(int target, int value);
void PutLL(int target, long long value);
```

Sending a message (non-blocking):

```
void Send(int target);
```

C++ API

Receiving a message (blocking, via buffers):

```
int Receive(int source);
```

If `source` is `-1`, it receives a message from any source.

Reading the message:

```
char GetChar(int source);
int GetInt(int source);
long long GetLL(int source);
```

Technical Restraints

- Why $M = 100$ nodes?
- How many messages can be sent?
- What is the time limit?

Technical Restraints

- Why $M = 100$ nodes?
  Large enough for a factor-of-M speedup to be noticeable.
  Small enough to have that many machines.
- How many messages can be sent?
- What is the time limit?

Technical Restraints

- Why $M = 100$ nodes?
  Large enough for a factor-of-M speedup to be noticeable.
  Small enough to have that many machines.
- How many messages can be sent?
  Sending a small message will take roughly 3-5ms. So we allow around 1000 messages and a few MB of messages per node.
- What is the time limit?

Technical Restraints

- Why $M = 100$ nodes?
  Large enough for a factor-of-M speedup to be noticeable.
  Small enough to have that many machines.
- How many messages can be sent?
  Sending a small message will take roughly 3-5ms. So we allow around 1000 messages and a few MB of messages per node.
- What is the time limit?
  Task statement contains a guarantee on the running time of input access methods (usually tens of nanoseconds). The total time limit is a few seconds (measured until all the processes terminate).

Basic Example

Problem: Compute the minimum of n integers ($n \leq 10^9$).

Basic Example

Problem: Compute the minimum of n integers ($n \leq 10^9$).

Functions available on all nodes:

```
int GetN();
int GetElem(int index);
```

Basic Example

Problem: Compute the minimum of n integers ($n \leq 10^9$).

Functions available on all nodes:

```
int GetN();
int GetElem(int index);
```

Inefficient single-node solution:

```
int n = GetN();
int minimum = INFTY;
for (int i = 0; i < n; ++i)
  minimum = min(minimum, GetElem(i));
cout << minimum << endl;
```

Solution

```
int n = GetN();
int minimum = INFTY;
for (int i = 0; i < n; ++i)
  minimum = min(minimum, GetElem(i));
cout << minimum << endl;
```

Solution

```
int M = NumberOfNodes(), no = MyNodeId(); // <--
int n = GetN();
int minimum = INFTY;
for (int i = 0; i < n; ++i)
  minimum = min(minimum, GetElem(i));
cout << minimum << endl;
```

Solution

```
int M = NumberOfNodes(), no = MyNodeId();
int n = GetN();
int minimum = INFTY;
for (int i = no; i < n; i += M) // <--
  minimum = min(minimum, GetElem(i));
cout << minimum << endl;
```

Solution

```
int M = NumberOfNodes(), no = MyNodeId();
int n = GetN();
int minimum = INFTY;
for (int i = no; i < n; i += M)
  minimum = min(minimum, GetElem(i));
PutInt(0, minimum); // <--
Send(0);
```

Solution

```
int M = NumberOfNodes(), no = MyNodeId();
int n = GetN();
int minimum = INFTY;
for (int i = no; i < n; i += M)
  minimum = min(minimum, GetElem(i));
PutInt(0, minimum);
Send(0);
if (no == 0) { // <--
  for (int p = 1; p < M; ++p) {
    Receive(p);
    minimum = min(minimum, GetInt(p));
  }
}
```

Proof of Concept

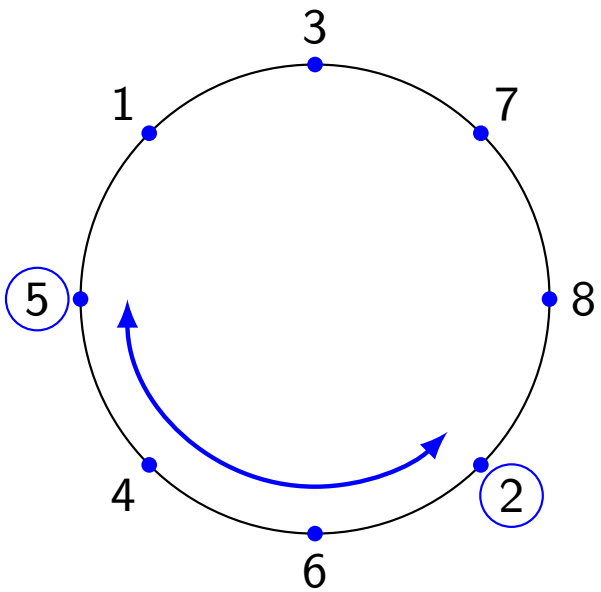
The real challenge in setting up a new competition type is in finding attractive problems:

- that challenge the contestants' creativity and problem-solving skill
- that do not require significant domain knowledge (in this case: knowledge of distributed programming paradigms)
- that are fun, but not tedious, to implement, once you have the correct set of ideas.

The tasks from two editions of Algorithmic Engagements and Distributed Track of Google Code Jam each show that there are plenty of such tasks!

Task "Workshop" (Algorithmic Engagements 2014)

A cycle of length $n \leq 10^9$ is specified using an oracle which for a given vertex returns its two neighbours. The neighbours are returned in an order that is not necessarily consistent with the order of the vertices on the cycle. A number of queries are given, each consisting of two vertices, and the task is to compute for each query the length of the shortest path between the two vertices.



Backend

Input generation requirements:

- Access to an arbitrary element.
- Consistency across nodes, and across accesses.
- Access times on the order of 100ns.
- Ability to serve data with total size on the order of 10GB or more.

Backend

Input generation requirements:

- Access to an arbitrary element.
- Consistency across nodes, and across accesses.
- Access times on the order of 100ns.
- Ability to serve data with total size on the order of 10GB or more.

The input data is generated on the fly.

Backend

Input generation requirements:

- Access to an arbitrary element.
- Consistency across nodes, and across accesses.
- Access times on the order of 100ns.
- Ability to serve data with total size on the order of 10GB or more.

The input data is generated on the fly.

The distributed and cloud computing aspect pops up also in the grading: one can rent virtual machines and pay by the minute!

Thank you!

Thank you for your attention!

(More task examples with solutions are in the paper!)