ALFA - a common concurrency framework for ALICE and FAIR experiments

Mohammad Al-Turany
GSI-IT/CERN-PH
This Talk

• Introduction
• Transport layer
• Deployment system
• Serializing message data
• Summary
• (Example)
ALFA

- Common layer for parallel processing.
- Common algorithms for data processing.
- Common treatment of conditions database.
- Common deployment and monitoring infrastructure.
ALFA

• Will rely on a data-flow based model (Message Queues).
• It will contain
  – Transport layer (based on: ZeroMQ, NanoMSG)
  – Configuration tools
  – Management and monitoring tools
• Provide unified access to configuration parameters and databases.
• It will include support for a heterogeneous and distributed computing system.
• Incorporate common data processing components
Design constrains

• Highly flexible:
  – different data paths should be modeled.

• Adaptive:
  – Sub-systems are continuously under development and improvement

• Should work for simulated and real data:
  – developing and debugging the algorithms

• It should support all possible hardware where the algorithms could run (CPU, GPU, FPGA)

• It has to **scale** to any size! With minimum or ideally no effort.
Before we start?

- Do we intend to separate online and offline? **NO**
- A message queue based system would:
  - Decouple producers from consumers.
  - Spread the work to be done over several processes and machines.
  - We can manage/upgrade/move around programs (processes) independently of each other.

April 1, 2014

M. Al-Turany, Annual Concurrency Forum meeting
Multi-processing vs. Multi-threading

- Different processes are insulated from each other by the OS, an error in one process cannot bring down another process.
- Inter-process communication can be used across network

- Error in one thread can bring down all the threads in the process.
- Inter-thread communication is fast
ALFA will use both: Multi-processing and Multi-threading
Correct balance between reliability and performance

• Multi-process concept with message queues for data exchange
  – Each "Task" is a separate process, which can be also multithreaded, and the data exchange between the different tasks is done via messages.
  – Different topologies of tasks that can be adapted to the problem itself, and the hardware capabilities.
This Talk

• Introduction
• Transport layer
• Deployment system
• Serializing message data
• Summary
• (Example)
What is available on the market and in the community?

• **RabbitMQ:**
  
  – One of the leading implementation of the AMQP protocol.
  
  – It implements a broker architecture
  
  – Advanced scenarios like routing, load balancing or persistent message queuing are supported in just a few lines of code.
  
  – **Less scalable and “slower”** because the central node adds latency and message envelopes are quite big.
What is available on the market and in the community?

• ZeroMQ
  – A very lightweight messaging system specially designed for high throughput/low latency scenarios
  – Zmq supports many advanced messaging scenarios but contrary to RabbitMQ, you’ll have combine the various pieces of the framework yourself (e.g: sockets and devices).
What is available on the market and in the community?

• ActiveMQ
  – is in the middle ground. Like Zmq, it can be deployed with both broker and P2P topologies.
  – Like RabbitMQ, it’s easier to implement advanced scenarios but usually at the cost of raw performance
ALFA will use ZMQ to connect different pieces together

- BSD sockets API
- Bindings for 30+ languages
- Lockless and Fast
- Automatic re-connection
- Multiplexed I/O
ZeroMQ is a New Library

April 1, 2014

M. Al-Turany, Annual Concurrency Forum meeting
Can be easily adapted by regular programmers!
Works across all types of networks

Network
Fastest way to make multithreaded apps that scale linearly to any number of cores
nanomsg is under development by the original author of ZeroMQ

- **Pluggable Transports:**
  - ZeroMQ has no formal API for adding new transports (Infiniband, WebSockets, etc). nanomsg defines such API, which simplifies implementation of new transports.

- **Zero-Copy:**
  - Better zero-copy support with RDMA and shared memory, which will improve transfer rates for larger data for inter-process communication.

- **Simpler interface:**
  - Simplifies some zmq concepts and API, for example, it no longer needs Context class.

- **Numerous other improvements, described here:** [http://nanomsg.org/documentation-zeromq.html](http://nanomsg.org/documentation-zeromq.html)

- **FairRoot is independent from the transport library**
  - Modular/Pluggable/Switchable transport libraries.

April 1, 2014  
M. Al-Turany, Annual Concurrency Forum meeting
ALFA has an abstract transport layer
Current Status

• The Framework delivers some components which can be connected to each other in order to construct a processing pipeline(s).

• All components share a common base called Device

• Devices are grouped by three categories:
  
  – **Source:**
    • Data Readers (Simulated, raw)
  
  – **Message-based Processor:**
    • Sink, Splitter, Merger, Buffer, Proxy
  
  – **Content-based Processor:**
    • Processor
This Talk

- Introduction
- Transport layer
- **Deployment system**
- Serializing message data
- Summary
- (Example)
How to deploy ALFA on a laptop, few PCs or a cluster?

• We need to utilize any RMS (Resource Management system)

• We should also run with out RMS

• Support different topologies and process dependencies
STORM is a very attractive option but no native support for C++!

Storm was designed from the ground up to be usable with any programming language. At the core of Storm is a Thrift definition for defining and submitting topologies. Since Thrift can be used in any language, topologies can be defined and submitted from any language.

Similarly, spouts and bolts can be defined in any language. Non-JVM spouts and bolts communicate to Storm over a JSON-based protocol over stdin/stdout. Adapters that implement this protocol exist for Ruby, Python, Javascript, Perl, and PHP. storm-starter has an example topology that implements one of the bolts in Python.

http://storm.incubator.apache.org/about/multi-language.html
How to deploy ALFA on a laptop, few PCs or a cluster?

• We have to develop a dynamic deployment system (DDS):
  – An independent set of utilities and interfaces, which provide a dynamic distribution of different user processes by any given topology on any RMS.
The Dynamic Deployment System (DDS) Should:

• Deploy task or set of tasks
• Use (utilize) any RMS (Slurm, Grid Engine, ... ),
• Secure execution of nodes (watchdog),
• Support different topologies and task dependencies
• Support a central log engine
• ....

See Talk by Anar Manafov on Alice Offline week (March 2014)
https://indico.cern.ch/event/305441/
This Talk

- Introduction
- Transport layer
- Deployment system
- Serializing message data
- Summary
- (Example)
Protocol buffers

• Google Protocol Buffers support is now implemented
  – Example in Tutorial 3 in FairRoot.

• To use protobuf, run cmake as follows:
  – cmake -DUSE_PROTOBUF=1
Boost serialization

- Code portability - depend only on ANSI C++ facilities.
- Code economy - exploit features of C++ such as RTTI, templates, and multiple inheritance, etc. where appropriate to make code shorter and simpler to use.
- Independent versioning for each class definition. That is, when a class definition changed, older files can still be imported to the new version of the class.
- Deep pointer save and restore. That is, save and restore of pointers saves and restores the data pointed to.

http://www.boost.org/doc/libs/1_55_0/libs/serialization/doc/index.html

This is used already by the CBM Online group in Frankfurt and to we need it to exchange data with them!
Protobuf, Boost or Manual serialization?

• Boost:
  – we are generic in the tasks but intrusive in the data classes (digi, hit, timestamp)

• Manual and Protobuf
  – we are generic in the class but intrusive in the tasks (need to fill/access payloads from class with set/get x, y, z etc).

Manual method is still the fastest, protobuf is 20% slower and boost is 30% slower.
Summary

• The AlFA project is starting

• The Communication layer is ready to use

• Different data serialization methods are available

• DDS and Topology parser are under development
Backup and Discussion
Integrating the existing software:

ROOT Files, Lmd Files, Remote event server, ...

FairRootManager

FairRunAna

FairTasks
  Init()
  Re-Init()
  Exec()
  Finish()

FairMQProcessorTask
  Init()
  Re-Init()
  Exec()
  Finish()

Root (Event loop)

ZeroMQ

April 1, 2014

M. Al-Turany, Annual Concurrency Forum meeting
FairRoot: Example 3

4 - Tracking stations with a dipole field

Simulation:
10k event: 300 Protons/ev

Digitization

Reconstruction:
Hit/Cluster Finder
2 x 2.4 Xeon Quad core Intel Xeon
16 GB Memory

Throughput ~ 1000 ev/s
Total Memory 263 MByte

100 s
263 MByte
2 x 2.4 Xeon Quad core Intel Xeon
16 GB Memory

171 s
6 * 263 MByte

Throughput ~ 3500 ev/s
Total Memory: 1578 MByte

Wall time: 171 s
Total Event: 60k events
2 x 2.4 Xeon Quad core Intel Xeon
16 GB Memory

Throughput $\sim 2660$ ev/s
Total Memory: 2104 MByte

Wall time: 300 s
Total Event: 80k events
2 x 2.4 Xeon Quad core Intel Xeon
16 GB Memory

Throughput Event/s

<table>
<thead>
<tr>
<th>Event/s</th>
<th>Throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1000</td>
</tr>
<tr>
<td>2</td>
<td>2000</td>
</tr>
<tr>
<td>4</td>
<td>3000</td>
</tr>
<tr>
<td>6</td>
<td>3500</td>
</tr>
<tr>
<td>8</td>
<td>2500</td>
</tr>
</tbody>
</table>
Before we continue:

1. **Sampler**
   - Process that read from ROOT files and send each entry as a message

2. **Proxy**
   - Bind on Input and Output

3. **Sink**
   - Get payloads, save/convert to ROOT Objects (TClonesArrays) then to files

4. **Processor**
   - Device class that contains the FairTask
2 x 2.4 Xeon Quad core Intel Xeon
16 GB Memory

Throughput ~ 7400 ev/s
Total Memory 1355 MByte

Wall time: 26.1 s
Total Event: 20k events

April 1, 2014

M. Al-Turany, Annual Concurrency Forum meeting
2 x 2.4 Xeon Quad core Intel Xeon
16 GB Memory

Throughput \sim 10990 \, \text{ev/s}
Total Memory: 1308 \, \text{MByte}

Wall time: 36.4 \, \text{s}
Total Event: 40k \, \text{events}
In Summary:

- **embarrassingly parallel**
  - 6-processes
  - Throughput ~ 1000 ev/s
  - Total Memory: 263 MByte

- **embarrassingly parallel**
  - 8-processes
  - Throughput ~ 3500 ev/s
  - Total Memory: 1578 MByte

- **Concurrent**
  - 4 processes
  - 2 file readers
  - 2 file sinks
  - Throughput ~ 2660 ev/s
  - Total Memory: 2104 MByte

- **Concurrent**
  - 4 processes
  - Separated reader
  - Throughput ~ 7400 ev/s
  - Total Memory: 1355 MByte

- **Concurrent**
  - 4 processes
  - Separated reader
  - Throughput ~ 10990 ev/s
  - Total Memory: 1308 MByte
The built-in core ØMQ patterns are:

- **Request-reply**, which connects a set of clients to a set of services. *(remote procedure call and task distribution pattern)*
- **Publish-subscribe**, which connects a set of publishers to a set of subscribers. *(data distribution pattern)*
- **Pipeline**, which connects nodes in a fan-out / fan-in pattern that can have multiple steps, and loops. *(Parallel task distribution and collection pattern)*
- **Exclusive pair**, which connect two sockets exclusively

---

M. Al-Turany, Annual Concurrency Forum meeting