The new inter process communication middle-ware for the
ATLAS Trigger and Data Acquisition system

Serguei Kolos, University of California Irvine, USA
Reiner Hauser, Michigan State University, USA
on behalf of the ATLAS TDAQ Collaboration

1. The ATLAS TDAQ Online Software

The ATLAS Trigger & Data Acquisition (TDAQ) project started almost twenty years ago with the aim of providing a scalable distributed data collection system for the experiment. While the software dealing with physics data flow was implemented by directly using low-level communication protocols, like TCP/IP, the control and monitoring infrastructure services for the TDAQ system were implemented on top of the CORBA communication middleware.

The ATLAS TDAQ Online Software places very demanding requirements on the Inter-Process Communication (IPC) system:
• The Online Software has to control and monitor the TDAQ system, which is composed of O(1)K processes distributed over O(1)K computers, connected via high-speed LAN.
• All components of the TDAQ system have to be operated with quasi-realtime performance as this is crucial for maximizing the efficiency of the experiment.

The first incarnation of the Online System, which was born in 1998, is based on the CORBA communication middleware. Two CORBA implementations have been used: JacORB for Java and omniORB for C++. They both satisfied the performance and scalability requirements and simplified development and maintenance of the Online Software.

However after more than 10 years successful experience with the CORBA brokers, we have decided the time is right to explore if there are new products on the IPC software market, which can improve our system performance and maintainability.

2. CORBA in the modern software world

CORBA is an open standard for distributed object computing, which was proposed in 1991 by the Object Management Group (OMG). This was the first attempt to provide a broad high-level standard for information exchange in a distributed software environment, which was quite successful and played an important role in the overall evolution of distributed software systems.

However many key features of the CORBA standard have a number of built-in drawbacks, which have become more and more prominent in recent years, making CORBA less attractive for modern software development:
• CORBA defines the PL-neutral Interface Definition Language for communication protocol description. The code for a specific PL is generated from such a description.
• Mapping of IDL to different PLs (e.g. C++ and Java) is old-fashioned and inefficient as it does not support zero-copy data transfer.
• CORBA provides a high-level object-oriented API which hides all aspects of the communication implementation.
• CORBA standardizes an API for object creation, registration, location and activation, thus assuring source code compatibility between different CORBA implementations.
• CORBA is quite complex and provides a lot of features, which are rarely or never used. In practice the source code compatibility works only for Java, while for C++ same or less complex changes are always required.
• CORBA compliant brokers can freely interoperate with each other. All CORBA implementations use the same data exchange protocol, which has some overhead in terms of both data size and processing time, mostly due to data alignment requirements of the CORBA network exchange protocol.

Tools like CORBA headers are quite complex and not optimized for efficiency.

3. The modern trends in the IPC software market

Fast evolution of computing systems and network technologies brought into life a huge number of different software communication systems. They are varying in many key aspects, including communication mechanism, implementation level, supported programming languages, communication protocols and so on. Some of those systems are applicable only for specific domains while others are general purpose systems, which can be used for implementing any possible type of communication.

There are three main aspects involved in distributed system development and the modern tendency is to handle them independently using different software:
• Marshalling libraries:
  • Java: MessagePack, Google ProtocolBuf, etc.
  • Communication libraries:
    • Boost: ASIO, ZeroMQ, RMQ, etc.
• Thread Management:
  • Boost: Thread, C++11 STD library, Intel Thread Building Blocks library, etc.

The absence of a language neutral interface description is compensated by modern improvements in programming languages like for example C++11.

4. Using ZeroMQ for Network Transport

ZeroMQ is a low-level C-style library for reading/writing something which looks like plain old sockets. All state management and error handling complexities are hidden.

ZeroMQ supports most of the widely used Programming Languages.

Fast evolution of computing systems and network technologies brought into life a huge number of different software communication systems. They are varying in many key aspects, including communication mechanism, implementation level, supported programming languages, communication protocols and so on. Some of those systems are applicable only for specific domains while others are general purpose systems, which can be used for implementing any possible type of communication.

There are three main aspects involved in distributed system development and the modern tendency is to handle them independently using different software:

• Marshalling libraries:
  • Java: MessagePack, Google ProtocolBuf, etc.
  • Communication libraries:
    • Boost: ASIO, ZeroMQ, RMQ, etc.
• Thread Management:
  • Boost: Thread, C++11 STD library, Intel Thread Building Blocks library, etc.

The absence of a language neutral interface description is compensated by modern improvements in programming languages like for example C++11.

5. ZeroMQ Performance and Scalability

To understand the performance and scalability of ZeroMQ several tests have been performed. For comparison we have been using the omniORB CORBA broker and the ICE framework provided by the ZeroC company.

The latter one is a modern CORBA-like object-oriented RPC framework, which is free of many CORBA drawbacks, so it’s an interesting candidate for performance comparison.

The hardware configuration used in testing:
Server: Linux running on the same computer: Intel Xeon E5645 4 cores 2.4 GHz, 4 GB RAM, 10Gbps Ethernet
Clients were equally distributed over 100 computers: Intel Xeon E5420 2.5 GHz, 16 GB RAM, 10Gbps Ethernet

All servers used the following configuration: 1 IO thread, 20 worker threads.
Clients send 1-byte string to the server and receive the same string back as fast as they can.

Test results:
The plots on the right present the test results. The first one shows average time for one request for all three systems and different number of concurrent clients. The second plot shows the number of CPU cycles the server spent to process a single request. The plots show all systems have excellent scalability and offer very good performance. ZeroMQ spends a lot more real time for a single request, probably because in the way requests are handled on the client side is different compared with the other two systems.

OMQ does not read data to the socket from the user thread, but just places the data into a queue which is handled by another dedicated thread. This thread reads the data from the queue and sends it to the socket.

At the same time OMQ shows significantly smaller CPU usage at the server side due to a very small overhead compared with the complex frameworks implemented by ICE and omniORB.

6. Conclusion

The CORBA standard has a long and successful history, but standard boundaries have been breached for the development community has shifted away from universal frameworks to a flexible combination of small independent libraries.

Using this approach for distributed systems implementation gives a number of advantages compared with the utilisation of a traditional high-level object-oriented framework:
• Performance – the network transport library provides a very low and efficient interface to the system.
• Simplicity – the API is very simple yet powerful.
• Flexibility – one can use or re-use any functionality which is required by the specific application thus avoiding unnecessary overhead.

We are now considering OMQ as the main candidate for the replacement of CORBA within the ATLAS TDAQ Online Software.