ATLAS Event Filter tests on the ASGARD cluster at ETH Zürich

ATL-DAQ-2001-007
12 Sep 2001

C. Bee, F. Etienne, E. Fede, C. Meessen, R. Nacasch, Z. Qian, F. Touchard

Centre de Physique des Particules de Marseille, Case 907, 163 avenue de Luminy,
F−13288 Marseille Cedex 9, France

Abstract

We present here results from a series of tests of the supervision system of the PC−based ATLAS Event Filter prototype performed at the *asgard* Cluster at ETH Zürich. Preliminary conclusions are also drawn on the operation of the name server used in the farm. Further tests will be necessary in order to fully understand effects which have not been seen on smaller configurations.

Version : 1.1

Reference : http://atlas.web.cern.ch/Atlas/GROUPS/DAQTRIG/EF/documents/ETHZ−jul01/ethz.pdf

Table of Contents

1 Introduction

We report here results obtained at ETH Zürich in July 2001 while testing the Supervision of the PC−based Event Filter prototype for the ATLAS experiment.

2 Material conditions of the tests

The tests were performed at ETH in Zürich, on the Beowulf *asgard* cluster [1]. The entire cluster was made available to us for a period of two days, although the testing time itself represented about 12 hours of working time for two persons (not including an overnight stability test).

2.1 The *asgard* Cluster

The *asgard* cluster comprises 240 bi−processor machines. The processors are Pentium III 500 MHz (for 192 nodes) or 600 MHz (for 48 nodes). Each machine has 1 Gbyte of RAM and a 6 GB IDE hard disk. Frames of 24 machines are linked by 100 Mb/s Ethernet switches and between the frames by a 1 Gb/s Ethernet switch. Three login servers and two file servers (Pentium III 500 MHz) are connected to the 10 frames via a 1 Gb/s Ethernet switch.

All machines run Red Hat Linux 6.2 (except the file servers which run SuSE Linux 6.3).

2.2 Event Filter architecture

2.2.1 Dataflow

The Version 4 [3] dataflow code (written in C++) was used. It follows the design described in [2].

The Event Filter farm comprises one or several sub−farms connected to the main DAQ data flow via the "Sub−Farm Input" (SFI) and "Sub−Farm Output" (SFO).

Inside a sub−farm, events are distributed to the processing nodes according to an "event type" encoded in the event header. A first process "D1" performs this sorting operation. Events of a given type are stored by a "D2" process from which they are extracted by event processing clients (processing tasks − PT), optionally via local "D3" buffers. Selected events are collected together in "C2" independent of their type. A local buffer "C1" may optionally buffer them before C2. Figure 1 summarises the dataflow tree.

All components are linked on a client−server basis. D1 (C2) is a server for both SFI (SFO) and D2. D2, D3 and C1 are client in input and server in output. PT are clients in both input and output. A name server allows a dynamic configuration of the tree. It has been decided to keep the implementation of this server as light as possible so as not to overload the dataflow operation.

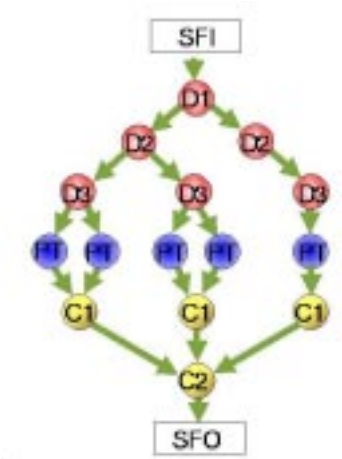

Figure 1: Dataflow tree

2.2.2 Supervision

The Supervision code is described in [5] and follows the design described in [4]. It is written in Java (JDK 1.3 toolkit).

A single Supervisor controls the whole farm. Java mobile agents are sent by the Supervisor to the different nodes of the farm where a mobile agent server is running. Agents can migrate from server to server, carrying their state with them. After having moved into a server, an agent becomes a local user and it can do everything that a local user can do, such as creating or deleting processes, collecting system or user information. Return status and collected information is then reported back to the Supervisor (Figure 2).

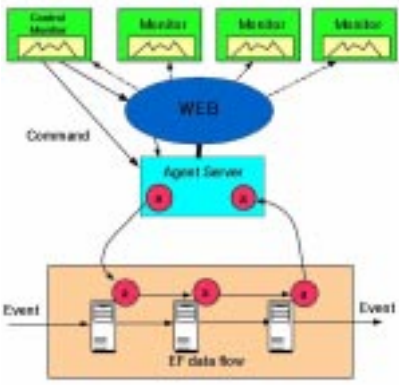

Figure 2: Supervision principle

A single interface has been built. Different instances provide a single control interface and multiple monitoring facilities.

3 Tests

The various farm configurations used for the following tests were described in an XML file described in [6]. The usage of such files, combined with interfaces written in Tk/Tcl,

provides the flexibility to easily handle very large configurations.

The first series of tests we have performed consisted of measuring the time necessary to handle a configuration on the farm, i.e. creating, visiting and deleting all the required processes for the dataflow. Preliminary operations (not included in the measured time) consisted of downloading the software on the local disk of every machine and starting the local Voyager daemons (also known as Mobile Agent Servers). Time was measured with a simple chronometer using the Supervisor windows reporting the agent activity to visually detect the end of the operation. Such a measurement technique is rather crude as uncertainties are introduced because the network activity may introduce delays while updating the display. Errors of the order of 5 seconds should be associated to the measurements, representing the fluctuations observed while performing several times the same operations.

3.1 Influence of the farm granularity

All available nodes (240) were used. We have measured the time to launch a configuration comprising a single sub−farm and an other one made of 20 sub−farms each having 12 compute nodes. 10 mobile agents (each one responsible for visiting 24 nodes) were launched in parallel. In both cases, the same total time (~ 11 s) was measured, from which we conclude that the farm granularity has no significant influence on the time taken to launch a particular configuration.

3.2 Influence of the number of mobile agents

The number of mobile agents sent in parallel to perform the supervision tasks can be easily modified. Each agent has a list of nodes to visit, therefore parallelising to some extent the farm supervision. We had already noticed on smaller configurations that the number of agents working in parallel had an optimum value which is a balance between the parallel execution of the operations and the congestion generated by the many reports simultaneously sent back to the supervisor. We have measured the time necessary to perform the 3 basic supervision operations *configure*, *visit* and *shutdown*, respectively: create all processes; visit all of them to collect and report information; delete all processes, as a function of the number of mobile agents for a single farm of 240 compute nodes. Results are displayed in Figure 3.

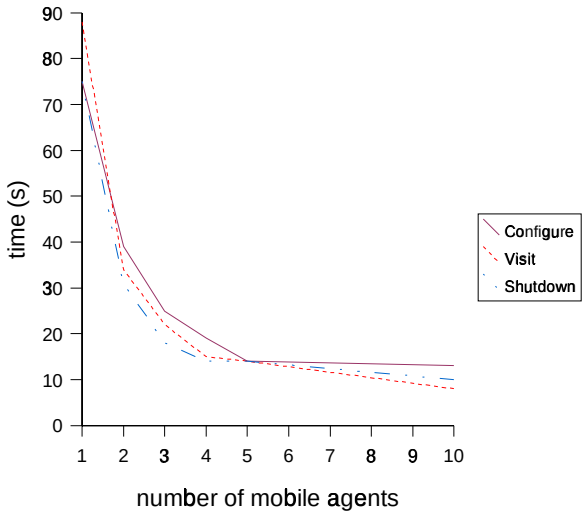

Figure 3: Time to configure, shutdown and visit as a function of the number of mobile agents. Measurement technique is rather crude and errors of the order of 5 seconds should be considered.

While performing these measurements, it was found that several processes did not start properly when the number of mobile agents was larger than a threshold value equal to 11. The number of missing processes as well as their location was apparently random. It was noticed however that the D3 and C1 components (having server capability in input or output) were missing more often than processing tasks (which are only clients in input and output). This led us to investigate the behaviour of the naming service, which is a crucial component in the inter−process relationship.

3.3 Influence of the number of processes accessing the name server in parallel

In these tests, we used a configuration comprising 20 sub−farms with a varying number of processing tasks. We had therefore 20 instances of the D1, D2 and C2 processes and as many D3 and C1 as processing nodes, i.e. 240. Every mobile agent was responsible for 48 nodes (with 5 agents being sent in parallel).

The number of processing tasks was varied from 2 per node (leading to a total number of 1016 processes) up to 10 per node (2928 processes). The time to start the processes increased from 13 s to 40 s respectively. All processes were successfully launched in all cases.

The number of nodes visited by each mobile agent was then decreased to 40 nodes thereby increasing the total number of agents in the system This configuration change has the effect of increasing the degree of parallelism of the process creation and thereby increasing the number of processes needing to access the name server at a given moment.

When this was done, a variable number of processes were found not to have been started. This therefore suggests a resource contention problem in the name server.

3.4 Maximum number of mobile agents as a function of the number of sub−farms

The number of sub−farms in the farm was varied, while keeping the total number of processing nodes (240). For each configuration, we determined the maximum number of mobile agents operating in parallel for which all processes were successfully launched. Results are presented in Table 1.

Number of sub−farms	*Minimum number of visited nodes per agent*	*Maximum number of mobile agents operating in parallel*
1	Between 20 and 22	11
10	Between 30 and 35	7
20	Between 30 and 35	7
30	Between 40 and 45	5

Table 1: Maximum number of mobile agents as a function of the number of sub−farms

Note that as the number of sub−farms increases, the total number of server compo−nents (D1, D2, D3, C1, C2) also increases while the number of processing (client) tasks remains constant. The results in table 1 indicate that this increase in the number of server tasks has the effect of reducing the total number of agents which we are able to use in parallel while ensuring that all processes are successfully started.

3.5 Influence of the number of name servers operating for the farm

In all the tests described above, a single name server was used for all machines in the farm. The name server is characterised by the name of its host and the port used to com−municate with the clients. We modified the configuration file so that different name servers could be used by different sub−sets of the farm. We have determined in each case the maximum number of mobile agents able to operate in parallel. Results are presented in Table 2 , for a configuration of 30 sub−farms.

Number of name servers	*Maximum number of mobile agents*
Running on the same machine	
1	5
2	7
4	8
Running on different machines	
2	10

Table 2 : Maximum number of mobile agents as a function of the number of name servers

The number of mobile agents does not scale with the number of name servers if they are running on the same machine, while it does if they are run on different machines. It can therefore be concluded that the origin of this effect is the insufficiency of a resource common to all processes, e.g. CPU or network interface.

4 Conclusions and perspectives

4.1 Functionality

We have demonstrated the ability of the Supervisor to launch from scratch large configurations comprising approximately 3000 processes distributed on 240 machines, in approximately 30 s. This figure compares favourably with those obtained using the On-line Software Process Manager utility [7]. In order to facilitate the measurements and improve their accuracy, better tools to measure the execution time of the supervision operations should be used in future

The importance of continuing such large scale tests is evident from the results obtained here. Many features, especially those dealing with the name server, had not been observed in previous tests on smaller clusters.

We are extremely grateful to the Department of Physics of ETH Zürich, co-owner of the cluster, for having allowed us to use the entire *asgard* cluster at short notice. It will become increasingly necessary for us to have flexible access to a large scale cluster (having more than 500 processors) so that future analyses can be rapidly and efficiently performed. Access to a large cluster at CERN to help in performing such analyses would greatly facilitate the further development of the ATLAS Event Filter.

4.2 Name server

The role of the name server in the operation of the Event Filter Farm is crucial in order to benefit from a flexible configuration system. The algorithm used by processes accessing the name server is deliberately very simple: a request is sent to the name server to subscribe (server starting) or to get the identification of an existing server (client starting). If the process does not get the answer before a pre-defined time-out, it retries. After 3 retries, the process considers that the name server does not exist and exits.

This algorithm may well be too simplistic for application in large farms and thus give rise to some of the problems described above. This hypothesis will be further investigated in the near future. If confirmed, we must then clearly identify the cause of the bottleneck responsible for the time-out: CPU; network interface; implementation of the naming service; other? New studies including the measurements of the CPU and memory usage as well as of the network statistics are now being planned.

The relative balance between the time-out duration, the number of retries and the efficiency of error detection will be investigated. These studies will be included in a more general approach of the fault tolerance and error recovery problem in the ATLAS High Level Trigger system.

4.3 General conclusions

The operation of the EF Java supervision system on a large farm hosting processes connected on a client−server scheme have been studied. Problems not seen during operation on smaller farms have now been observed. New studies will be undertaken to analyse in more detail their origin and to suggest possible solutions.

5 Acknowledgements

We would like to thank the Department of Physics of ETH Zürich and in particular, Dr. C. Grab, Dr. A. Biland and G. Sigut, for the use of the *asgard* cluster and for their kind assistance before and during the tests.

6 References

[1] The asgard Cluster http://www.asgard.ethz.ch
[2] C. Bee et al. A High Level Design of the Sub−Farm Event Handler (1997) http://atddoc.cern.ch/Atlas/Notes/061/Note061−1.html
[3] C. Meessen et al. Event Filter Dataflow Software (2000) http://documents.cern.ch/cgi−bin/setlink?base=atlnot&categ=Note&id=daq−2001−001
[4] C. Bee et al. Event Handler Supervisor High Level Design (1999) http://atddoc.cern.ch/Atlas/Notes/Note092−1.html
[5] Z. Qian et al. PC−based Event Filter Supervisor : Design and Implementation (2000) http://documents.cern.ch/cgi−bin/setlink?base=atlnot&categ=Note&id=daq−2001−002
[6] http:////atlas.web.cern.ch/Atlas/GROUPS/DAQTRIG/EF/EFDB/index.html
[7] D. Burckart et al. Online Software Scalability Tests http://documents.cern.ch/cgi−bin/setlink?base=agenda&categ=a01288&id=a01288s11t1/transparencies