INCREASED ILC SOFTWARE PERFORMANCE USING CLOUD COMPUTING*

B. Krupa, T. Lesiak, T. Wojtoń, L. Zawiejski

on behalf of the FCAL Collaboration

The Henryk Niewodniczański Institute of Nuclear Physics
Polish Academy of Sciences
Radzikowskiego 152, 31-342 Kraków, Poland

(Received May 22, 2015)

This work presents the speed up measurements of getting data for study of $e^+e^- \to e^+e^-X$ process using Cloud Computing. The analysis contains performance measurements and general review on the dependency between the task size, the number of CPU cores and the time used to compute.

DOI:10.5506/APhysPolB.46.1337
PACS numbers: 89.20.Ff

1. Introduction

The physics simulation, reconstruction and analysis need the increasing amount of computing power and consequently more time. The computing models are evolving, trying to provide easy-to-use tools. The latest and most reliable model is Cloud Computing. This model has many facilities. In this work, the Cracow Cloud One platfrom [1, 2] was used.

The $e^+e^- \to e^+e^-X$ process [3] simulations for the ILC experiment need high statistics for better analysis, also input parameters for detector optimalization change frequently. This process takes a lot of time. The problem is to decrease the waiting time for data to be analyzed to minimum. Additionally, the measurements could be helpful at the final stage of data production.

2. Cloud Computing

Cloud Computing provide computer infrastructure on-demand to end-users. Additionally, the user can allocate resources and scale them when

* Presented at the Cracow Epiphany Conference on the Future High Energy Colliders, Kraków, Poland, January 8–10, 2015.

needed — this means that hardware is really used, because if someone needs more resources for computing — he/she just reserves more resources. Also, if the resources are no longer needed, the user can release them. There are a few models of Cloud Computing:

— Software as a Service (SaaS) — applications used by the users, for example — webmail.

— Platform as a Service (PaaS) — a set of tools, services and software ready to use.

— Infrastructure as a Service (IaaS) — a base for other models, this includes hardware, network, storage.

Especially for High Energy Physics computation, which needs a lot of resources for *e.g.* CPU cores and memory, a virtual cluster solution is proposed. It is a set of fully controlled and configured virtual machines, which are ready to use within minutes.

The Cracow Cloud One platform has the dedicated virtual clusters called Farms. The Farms are a configured set of virtual machines with an access to external storage for data which are easy to create and manage.

3. ILC software and virtual clusters

To demonstrate a real example and imagine the problem, 1000 events of $e^+e^- \to e^+e^-X$ process at 500 GeV centre-of-mass energy have been generated by PYTHIA [4] generator. The simulation and reconstruction of this data in a sequential way takes more than 10 hours. But if this case has been parallelized with 25 of CPU cores, the whole process takes 1 hour.

A typical processing chain of preparing data for analysis could be divided into 3 parts:

— events generation,

— simulation,

— reconstruction.

As mentioned before, the physics processes were generated by Pythia. The generated events are stored in ASCII files (HepEvt) and then simulated using Mokka [5] software. The reconstruction is done by Marlin [6] framework.

The use of virtual clusters requires the possibility of running more than one instance of Mokka and Marlin. For this reason, the whole process must be divided into smaller jobs.

The events are generated in a sequential mode without parallelization, because this would require changing the generator code. Otherwise, the generation takes only 0.1% of the total time.

Therefore, an event file is splitted into parts by specially developed python script. The script accepts as a parameter the number of parts to split the task.

Mokka and Marlin require some configuration files which contain a path to input and output data, and the number of events to processing.

These files are created from the prepared template and then the jobs could be sent to worker nodes. Automated checking will be done after the jobs are finished.

The whole process is automated and is shown in figure 1. The user provides only the necessary information in the startup script:

— number of events,

— number of parts,

— path to the storage directory.

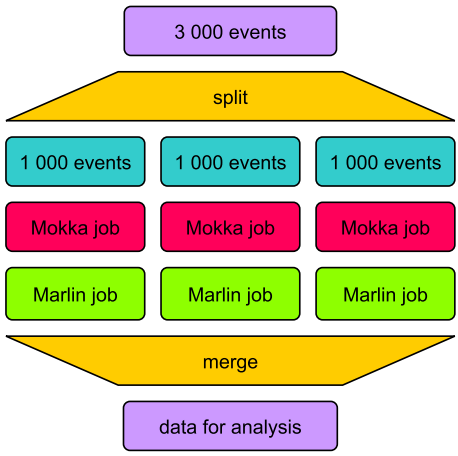

Fig. 1. Common scheme of automating processing of 3 000 events splitted into 3 jobs.

The execution of the startup script will split events to smaller parts and distribute then to worker nodes. After this procedure, Mokka and Marlin could run in parallel, when each instance of software processes one part of events. In the end, the parts are merged together to provide files for analysis.

4. Results

For testing purposes, the computation has been made for 100 000, 500 000 and 1 000 000 events. Each run was precisely measured and performance data was collected to evaluate parallel computing speed-up (Fig. 2).

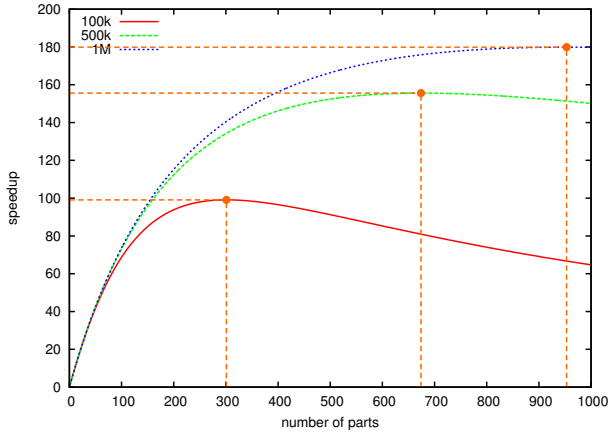

Fig. 2. Speed-up of 100 000, 500 000 and 1 000 000 events parallelization.

The time necessary to execute a single operation was estimated and shown in Table I.

TABLE I

Single operation time estimations.

	Type	Time [s]	Depend on
Generating	Seq.	0.036	event
Splitting	Seq.	0.0004	event
Preparing files	Seq.	0.0008	part
Mokka init	Seq.	38.5	part
Mokka compute	Par.	36	event
Marlin init	Seq.	4.8	part
Marlin compute	Par.	3.35	event
Merge	Seq.	0.1	event

The operations depend on the number of events or the number of parts. This means that an increasing number of parts relatively extends the time of processing. The operations were categorized into two types: sequential and parallel. Parallel operations could be executed simultaneously.

At this point, it is important to mention that Mokka and Marlin initialization time are treated as a sequential-only operation, because software needs to be connected to shared database. A simultaneous access could cause errors, thus a decision was made not to parallelize this operation. Therefore, here is some area to optimize. One of the possibilities is to create a local mirrored database.

The total time of the whole process from this measurements and combinations of the number of events and number of parts are evaluated.

CPU core usage over full computation time for 100k, 500k and 1M events is shown in figure 3. Respectively the shortest time for each dataset is correlated with the corresponding number of CPU cores. For dataset of 100k events, computing time is 11 h with 301 CPU cores usage (Table II). A further increase of the number of cores increases time. But decrease number of cores by factor 2 takes less than 2 hours more. In other words, the decreasing number of cores by factor 2 needs waiting for results about 10–15% longer.

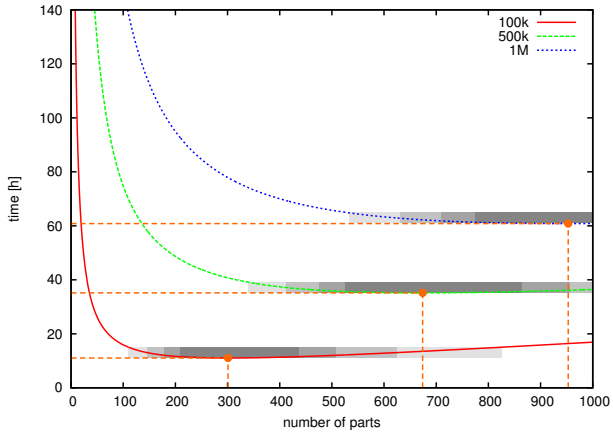

Fig. 3. Total time of processing 100 000, 500 000 and 1 000 000 events.

TABLE II

An example of the changing the computing time according to the number of the CPU cores.

Events	CPU	Time [h]
100k	301	11.0
500k	674	35.2
1M	953	60.8
100k	150	12.9 (11.0 + 17.3%)
500k	337	39.2 (35.2 + 11.4%)
1M	476	66.6 (60.8 + 9.5%)
100k	100	15.9 (11.0 + 44.5%)
500k	224	46.0 (35.2 + 30.7%)
1M	317	76.2 (60.8 + 25.3%)

5. Summary

The main conclusion of this work is that the number of CPU cores should be properly selected. Also processing a large amount of data should be preceded by some performance measurements to optimize the whole process. The described system could be used as a kind of benchmark to select how many events could be processed at a specific time with the defined number of CPU cores. Otherwise, it could be helpful to estimate how much time takes processing of a certain number of events with a specified number of CPU resources.

REFERENCES

[1] J. Chwastowski *et al.*, *Comput. Sci.* **13**, 103 (2012).
[2] Cloud Computing One website: `http://cc1.ifj.edu.pl`
[3] B. Krupa *et al.*, *Acta Phys. Pol. B* **46**, 1329 (2015), this issue.
[4] Pythia 6.4 Physics and Manual, `arXiv:hep-ph/0603175`.
[5] Mokka webpage: `http://mokka.in2p3.fr/`
[6] F. Gaede *et al.*, Marlin webpage:
`http://ilcsoft.desy.de/portal/software_packages/marlin/`