Single top quark production at the LHC: Data processing and cross section measurement
Lee, H.C.

Citation for published version (APA):
Lee, H. C. (2013). Single top quark production at the LHC: Data processing and cross section measurement
With the data prepared by the offline data processing, subsequently physics analyses are carried. Due to its small cross section, the search for single top quark production in the $Wt$-channel requires thorough analysis on large amounts of data. The throughput of the data analysis has impact not only on the time it takes to make the measurement, but also on the feasibility of repeating the analysis for improving the quality of the measurement.

Looking into the data analysis workflow of our search for single top quark production in the $Wt$-channel, the aim of this chapter is to obtain an optimal setting in computing for better data analysis throughput. In Section 5.1, we introduce the workflow of our user data analysis utilizing the D3PD datasets and discuss the computations required for a complete analysis. In Section 5.2 and 5.3, we analyze the timing information collected at runtime and compare the optimization of the data analysis throughput on two different computing systems: the WLCG and a system called “the cloud”.

### 5.1 The workflow

Our data analysis in the search for single top quark production in the $Wt$-channel uses the D3PD datasets, produced centrally by the top-quark research group in the ATLAS collaboration (referred hereafter as TopD3PD).

Figure 5.1 shows a schematic illustration of the TopD3PD format. The TopD3PD is a n-tuple with elements of event properties essential to the top-quark related studies. There are event-wise properties such as the detector run number and event id, and properties of reconstructed objects such as the four-momentum of final state particles. For the events generated by the Monte-Carlo simulation, properties of the generated
Chapter 5. Optimizing Throughput of User Data Analysis

Figure 5.1: Schematic illustration showing the layout and the contents of the TopD3PD n-tuple.

Figure 5.2: The data analysis workflow using the TopD3PD datasets.

particles (the truth information) are also presented. Each event is represented as a entry in the table of the n-tuple.

Our data analysis workflow on TopD3PD datasets is illustrated in Figure 5.2, and it consists of four subprocesses:

- **The EventLoop process** runs over all events in the TopD3PD datasets. The main purpose of this process is to apply the event selection criteria we have discussed in Section 3.2 to select signal-like events. In order to give flexibility for applying different object calibration algorithms, some calibrations discussed in Section 3.1 take place in this process. In addition, derived observables needed for physics studies in later steps are calculated here using the information available in the TopD3PD datasets. For example, the mass of a $W$ boson decaying into two quarks is reconstructed within the EventLoop process by summing up the four-momenta of two properly selected jet objects.

- **The histogram making process** runs over the signal-like events selected by the EventLoop process. It creates histograms to represent event distributions for physics observables, and it is our first examination on how well a physics process is described by simulation. Subsequently, our search will be narrowed down to an area in the phase space where the signal is more clear and significant (see
The workflow (Chapter 7).

- **The physics analysis process** consists of physics studies based on the event distribution histograms. They are studies of estimating the QCD and the $W^{+}$jets backgrounds from data (see Chapter 6), performing the measurement on the signal cross section and studying possible uncertainties on the measurement (see Chapter 7). In the case of the QCD and the $W^{+}$jets estimation from data, results are provided back to the histogram making process for another iteration to include the corrected QCD and $W^{+}$jets backgrounds.

For a complete analysis, the three processes have different characteristics in terms of number of iterations and the required computing power:

- **The EventLoop process** needs to be repeated if there are changes of event properties in the TopD3PD datasets. This happens each time the ATLAS or simulated data are reconstructed with improved algorithms. For evaluating the systematics, additional iterations are performed on the simulated events, each of them adjusts event properties to account a source of the systematics. Given the processing rate of 40 events per second on a single computer (to be discussed later in Figure 5.3), running one iteration over the full TopD3PD sample with 760 million events in total requires more than 200 CPU days of computing power. Taking into account a minimal number of 10 iterations on the simulated events for evaluating the systematics, the EventLoop process requires at least 3 CPU years for a complete data analysis.

- **The histogram making process** is repeated whenever the event distribution is modified following a new iteration of the EventLoop process or an update on background measurements, e.g. better estimates on the QCD multi-jet and $W^{+}$jets backgrounds. With the amount of events reduced by the EventLoop process, the total computing power needed for one round of this process is about few CPU days.

- **The physics analysis process** takes few CPU hours for one iteration. During the analysis, several interactive iterations are necessary to adjust settings of algorithms for optimizing the measurement and evaluating the uncertainties of the measurement.

As the majority of the computation is involved in the EventLoop process, the throughput of our user data analysis depends on how fast the EventLoop process can be done. Moreover, the faster the EventLoop process can be done, the more feasible it is to repeat the analysis to improve the quality of the measurement, by either fine-tuning the analysis method or including the latest reconstructed data. For those reasons, we will concentrate on the EventLoop process for the rest of this chapter and discuss how it’s performed on different computing systems.
5.1.1 The EventLoop program

The logical structure of the EventLoop program is illustrated as Algorithm 1. The program loops over event entries in the D3PD n-tuple. For each event, kinematic properties of physics objects are firstly adjusted accordingly for object calibration or systematic variations. Based on the adjusted properties, objects are selected. If the event fulfills the event selection criteria, calculations take place to derive additional observables needed for physics studies. Finally, object properties and physics observables of the selected events are written to a smaller n-tuple as output.

Algorithm 1 The pseudocode of the EventLoop program

1: repeat
2: read next event from in
3: adjust kinematic properties of physics objects by a
4: select physics objects by o
5: if s is fulfilled then
6: calculate additional observables
7: write observables and object properties to out
8: end if
9: until end of in

After profiling the EventLoop program, Figure 5.3 shows that the time spent on the event reading (i.e. line 2 in Algorithm 1) takes up to 43% of the total processing time. One notes that the profiling is done with D3PD events already present on the disk of the computing node. In a realistic case, the data event has to be staged from a remote storage by either transferring whole data file to the computer node before event reading, or performing the event reading over network, which adds an extra amount of time to the event reading. Thus, the performance of the EventLoop program is determined by the efficiency of the event reading.

Factors affecting throughput

To analyze the EventLoop throughput, we formulate it here in relation with key factors we will be measuring and discussing in the chapter.

Generally speaking, an EventLoop process is divided into multiple workloads, each of them runs the EventLoop program independently over an equal amount of events.
5.1. The workflow

Figure 5.3: Distributions of the total elapsed time for processing an event (line 2-8 of Algorithm 1), and the time spent on reading the event into the process (line 2 of Algorithm 1). Mean values of the distributions are taken as time measurements shown in the legend. Input events are read from the local disk of the computing node.

Figure 5.4: An EventLoop process split into 4 identical workloads executed in parallel by two identical processors. The total core time consumed by the whole process and the total elapsed time to finish the process are indicated accordingly as \( T_{\text{core}} \) and \( T_{\text{elapsed}} \) along the time line.
As illustrated in Figure 5.4, the elapsed time of the EventLoop process ($T_{\text{elapsed}}$) is related to the total core time consumption ($T_{\text{core}}$) and the number of processors being used in parallel ($N_{\text{proc}}$) as

$$T_{\text{elapsed}} \propto \frac{T_{\text{core}}}{N_{\text{proc}}} \quad (5.1)$$

With the elapsed time of the EventLoop process, the EventLoop throughput is then defined as

$$\text{throughput} = \frac{N_{\text{events}}}{T_{\text{elapsed}}} \quad (5.2)$$

where $N_{\text{events}}$ is the total number of events to be processed. Given $N_{\text{events}}$ a constant, the throughput is discussed in terms of $T_{\text{elapsed}}$ throughout this chapter.

Assuming the wall-clock time of executing workload $i$ is $t_{\text{wall}}^i$, the total core time consumption $T_{\text{core}}$ in Equation 5.1 is now written as

$$T_{\text{core}} = \sum_i t_{\text{wall}}^i \quad (5.3)$$

In a practical situation, system overhead and I/O latency experienced by workload $i$ take part of $t_{\text{wall}}^i$. For understanding their impact on the EventLoop throughput, we relate $t_{\text{wall}}^i$ further to the CPU utilization fraction, $r_i$, defined as

$$r_i = \frac{t_{\text{cpu}}^i}{t_{\text{wall}}^i} \quad (5.4)$$

where $t_{\text{cpu}}^i$ is the CPU time consumed by workload $i$.

**ROOT I/O optimization for event reading**

As mentioned earlier, event reading efficiency is essential to the EventLoop performance. Thus, it’s important to understand the data structure of the D3PD n-tuple and apply possible optimizations for event reading.

As illustrated in Figure 5.5, the D3PD n-tuple is represented by an object called “TTree” of the ROOT [133, 134] framework. The TTree object contains branches, each of them is associated with a n-tuple element to hold data of the corresponding event property. Each branch has a memory buffer called “basket”. The buffer has a fixed size of 32 kilobytes by default. When filling the event data into the n-tuple, event properties are firstly stored in their corresponding baskets. Data in a basket is then
5.1. The workflow

Figure 5.5: Schematic illustration showing how the properties of two events in a D3PD n-tuple are structured into a ROOT file. Each n-tuple element is associated with a TTree branch by which the data of event property is stored in a fixed-sized memory buffer called basket. Baskets are colored to indicate their associations with different n-tuple elements. Data in a basket is flushed out to the ROOT file when the basket is full, and hence they are drawn in unit of basket in the ROOT file with numbers indicating the events whose data is contained in the basket. The data structure in the ROOT file illustrated here shows a case that the baskets for properties \textit{el\_pt} and \textit{el\_E} are flushed twice before baskets for other properties are flushed out to the file.

flushed out to the ROOT file on disk whenever the basket is full. Optionally the data can be compressed before being written to disk.

Since the event properties can vary in size, branches are not filled up and flushed out to the file at the same time. This leads to a situation that the properties of a single event may not be placed continuously in the ROOT file. Therefore, reading the event data requires multiple seek and read operations on the file, which turns out to be inefficient.

For addressing the inefficiency, ROOT introduces a so-called “TTreeCache” that manages a memory buffer for caching the event data read from the file. Details about how TTreeCache plays a role in optimizing the ROOT I/O performance are discussed in [135]. Essentially, it allows ROOT to plan ahead and fetch more event data in a system’s \texttt{read()} call on the file. With more event data already presented in the memory cache, following requests of event data can then be served from memory rather than doing seek and read operations on the file.
Chapter 5. Optimizing Throughput of User Data Analysis

The optimal size of the TTreeCache depends on the content of the D3PD n-tuple and the way data is read into the analysis. To determine the optimal size for our EventLoop program, a simplified test is conducted to perform only the event reading (i.e. skipping line 3-8 of Algorithm 1) from a TopD3PD file consisting of 10,000 physics events. We also compare the event reading on local disk with the reading over network.

The event reading rate in relation to the TTreeCache size is shown in Figure 5.6. As suggested [135], event reading over network benefits largely from having the TTreeCache enabled. Up to the size of 100 megabytes, the event reading rate increases promingly while enlarging the TTreeCache size. The improvement becomes uncertain or rather marginal if going further to more than 100 megabytes. Based on the observation, we set 100 megabytes as our optimal TTreeCache size.

![Figure 5.6](image)

**Figure 5.6:** Rate of reading events from a TopD3PD file consisting of 10000 events. The rate is measured with different configurations of the TTreeCache size. Event reading on local disk is compared with the reading over network using storages and protocols provided by the data centers at SARA, NIKHEF and CERN.

We also observed that the overall network traffic generated by the event reading is reduced by enabling the TTreeCache. Figure 5.7 shows the network traffic during two tests of reading a 1-gigabyte TopD3PD file 10 times. The total amount of data being read over the network can be seen as the area under the histograms. It is about 40 gigabytes when the TTreeCache is disabled (the histogram on the left), 4 times larger than necessary. This is due to the fact that ROOT intrinsically reads data from file in unit of basket. When an event property is requested, ROOT actually reads in the entire basket from file in which the property is stored. Without the TTreeCache, same basket has to be read several times from file if event properties are requested by the analysis in a random order. Enabling the TTreeCache prevents multiple reading on the same basket and hence reduces the network traffic. On Figure 5.7, the histogram on the right shows the network traffic when the TTreeCache is enabled. The area under the histogram is about 10 gigabytes which is close to the amount of data one actually
needs for the test.

![Network Traffic](image)

**Figure 5.7:** Network traffic generated by two tests of reading a 1-gigabyte TopD3PD file 10 times over network. The time axis shows the clock time on the day when the two tests were conducted.

### 5.2 Data Analysis on the Grid

As suggested by Equation 5.1, the more processors \( N_{\text{proc}} \) we can mobilize in parallel, the shorter the EventLoop elapsed time. Although the majority of the resources on the WLCG are taken by the offline data processing, there is still a considerable amount of resources available for user data analysis. To benefit from this large amount of computing resources, our primary approach is run the EventLoop program on the WLCG.

As we have mentioned earlier, an EventLoop process consists of multiple workloads. The execution of workloads on the WLCG are carried out by the grid jobs.

#### 5.2.1 The user toolkits

Unlike the offline data processing that is centrally managed and operated, data analyzers are responsible for managing the workloads as well as the data themselves on the grid since the analysis program is mostly developed for a specific study.

On the other hand, user analysis on the grid can benefit from the achieved efficiency in data and workload management by taking advantage of the development and operational effort investigated for the offline data processing. As a result, users are
provided with two toolkits, pathena and Ganga, that are integrated with the data and workload management systems developed for the offline data processing.

The two toolkits distinguish themselves from each other by their design principle, featuring different approaches of workload management for users to chose. An overview of possible ways to manage data analysis workloads using Ganga and pathena are shown in Figure 5.8. These two toolkits serves as interfaces to various workload management systems, enabling data analyzers to utilize distributed resources provided by the WLCG or a local computer cluster.

**Figure 5.8:** Overview of possible ways to manage data analysis workloads on distributed computing resources using the two user toolkits: pathena and Ganga. The arrow indicates the flow of the workload submission.

**User toolkit: pathena**

The pathena toolkit is designed explicitly to performe data analysis workloads through the PanDA system. It is developed as a light-weight client consisting of a set of command-line tools allowing users to interact with the PanDA system for workload submission and bookkeeping.

The tool for workload submission translates user arguments, such as user algorithms, input and output datasets, into analysis workloads and sends them to the PanDA server together with sandboxes in which user codes are presented. Each workload
is given by the PanDA server an unique ID as the reference for bookkeeping. The interface of the workload submission tool is customized in such that the usage of it is similar to running the data analysis interactively on a desktop computer. With the similarity, the tool can be adopted quickly to migrate analysis workloads from a desktop computer to the grid.

With the workload ID, one can use the bookkeeping tool to retrieve the workload status or to cancel the workload in the PanDA system if needed. The bookkeeping tool has an interactive interface with scripting capability for performing workload management in batch.

**User toolkit: Ganga**

Different to pathena, the design of Ganga [136] aims at providing a general and extensible end-user framework for managing workloads on various distributed systems. With the object-oriented design, Ganga defines a computing task\(^1\) as a composition of six basic abstract objects as illustrated in Figure 5.9.

\[\text{Figure 5.9: Abstract objects of a computing task in Ganga and their example implementations (plugins) supporting the ATLAS data analysis.}\]

The “Application” object describes the actual computation (i.e. the workload). Inputs and outputs of the workload are defined respectively by the “InputData” and the “OutputData” objects. The “Backend” object specifies the computing system on which the task is executed. The “Splitter” and “Merger” objects are used to specify

\[^1\]Since Ganga was originally designed to interface with the batch system of a computer cluster, a computing task is actually called “job” in Ganga. It should not be confused with the “grid job” we are referring to in this thesis, which is the one performs the workload execution on a computer node.
mechanisms for parallelizing the task into workloads and for combining outputs into final results. Concrete implementations of those abstract objects are plugins. As the implementation follows a well-defined schema and interface, plugins of the same abstract object are highly inter-changeable. By exploiting this feature, Ganga offers possibilities to migrate data analysis computations from one system to another by replacing the “Backend” plugin interfacing with a different workload manager.

GANGA provides a simple but flexible programming interface allowing users to create, manipulate and manage workloads programmatically in scripts or interactively through a command-line interface. Bookkeeping and routine works of management such as status checking, output downloading, failure recovering are handled automatically by the core of the framework.

5.2.2 Efficiency of global user data analysis

Within the ATLAS collaboration, many data analyses are performed on the WLCG. According to the statistics [137], more than 1,500 users have run their data analysis workloads on the WLCG and the average data analysis throughput is approximately 366,000 workloads per day. Figure 5.10 indicates that user data analysis utilized about 22% of the grid resources in 2011 in terms of the wall-clock time consumption. It is interesting to note that this fraction is not too far from the 29% predicted by the ATLAS computing technical design report in 2005.

Figure 5.11(a) shows that the success rate is about 77% for the grid jobs running the data analysis workloads. A breakdown of failures is given in Figure 5.11(b). Unlike the offline data processing (Figure 4.12), the runtime failure becomes the dominant issue. It can be understood as the fact that user analysis codes are usually developed on a specific computing system; they may not be properly tested at the first place to cope the heterogeneity of the grid resources. Despite the failures coming from the user analysis codes, a study [137] showed that only 13% of the failures can be successfully retried. To help users deal with the rest of failures, a discussion forum is setup to establish the communication between the users, the experts (i.e. experienced users), and the operational shifters.

Similar to the offline data processing, grid sites are also tested regularly by the HammerCloud system [130] to protect users against temporarily malfunctioning sites. Sites failing any functional tests on two or more successive runs are temporarily excluded so that no further analysis workloads will be brokered to the sites.

---

2The prediction in the ATLAS computing technical design report assumed 600 active analysis users, each of them requires an average of 15 KSI2k CPU resource concerning possible user analysis scenarios. With the assumption of 60% analysis efficiency, it adds up to totally 15 MSI2k of the CPU resource consumption for user analysis, which is equivalent to 29.6% of the predicted 50.6 MSI2k for all ATLAS grid computing activities summarized in Table 7.2 of the report.
5.2. Data Analysis on the Grid

Figure 5.10: The total wall-clock time consumption on the WLCG in year 2011 split into the offline data processing and the user data analysis activities. Other computing activities such as software validation are ignored here as they consumed negligible amount of wall-clock time.

Figure 5.11: Statistics of the grid jobs running the data analysis workloads in 2011: (a) the fractions of completed grid jobs in different status, and (b) the categorization of failures.
5.2.3 Throughput of the EventLoop process

Although the overall throughput and success rate of the global user analyses on the grid is promising, our main interest is what it implies for the throughput of the EventLoop process.

To answer this question, we set up a study to analyze the throughput of an EventLoop task running over the simulated events of the $tt$ process. The task is split into 271 workloads, each of them processes 60,000 events. To simulate the real situation during the data analysis lifecycle, the same task is repeated several times to investigate the effect coming from the dynamics of the grid system.

In a heterogeneous system like the grid, the time spent on executing the workload can vary depending on the resource capacity. Figure 5.12 shows the distributions of the CPU time and wall-clock time of the workloads running on the WLCG. The CPU time consumption is closely related to the processor capacity. According to the means and the standard deviations of the distributions, one sees that the variation of the CPU time consumption is about 15% of the mean value indicating that the effect from the processor capacity is not significant.

![Figure 5.12](image)

*Figure 5.12: Distributions of the CPU and wall-clock time of the EventLoop workloads running over 10,000 events on the WLCG. The mean values and the standard deviations are shown in the legend.*

On the other hand, the deviation on wall-clock time is about 40% of the mean value suggesting that the event reading performance and other system overhead are widely different between grid sites. This observation is supported by the CPU utilization fraction shown in Figure 5.13 with relation to grid sites. Not only the mean value ranges from 0.48 to 0.88 across grid sites, the error bars also indicates that large deviation can happen at the same grid site.
5.2. Data Analysis on the Grid

We also observed that widely varying CPU utilization fraction has direct impact on \( T_{\text{core}} \). Measured from 20 runs of the same EventLoop task, Figure 5.14(a) shows a trend that the higher the CPU utilization fraction, the lower \( T_{\text{core}} \) the task consumes. On the other hand, the expected correlation between \( T_{\text{core}} \) and \( T_{\text{elapsed}} \) in Equation 5.1 with an assumption of a constant \( N_{\text{proc}} \) is not observed, as we see in Figure 5.14(b) that \( T_{\text{elapsed}} \) is rather independent to the CPU utilization fraction or even largely uncertain with a given CPU utilization fraction. We need to inspect the profile of \( N_{\text{proc}} \) to understand its impact on the throughput.

On the grid, \( N_{\text{proc}} \) is in fact not a constant value as the resource available for processing workloads changes dynamically. Therefore it can only be assessed alternatively by looking into the evolution of the number of running workloads. In Figure 5.15, the number of running workloads as well as the accumulated number of finished workloads are shown along the timeline of the task execution. Time 0 is taken as the moment the first workload is submitted to the PanDA system. The two sub-figures correspond to two runs of the same task started at different moment in time.

Comparing the maximum \( N_{\text{proc}} \), the run in Figure 5.15(a) managed to reach as many
Figure 5.14: Measurements of (a) $T_{\text{core}}$ and (b) $T_{\text{elapsed}}$ from 20 runs of the same EventLoop task, as a function of the averaged CPU utilization fraction taken as the mean of the CPU utilization fractions of the workloads in a run.

Figure 5.15: Time evolutions of the number of running workloads and the accumulated number of finished workloads from two runs of the same EventLoop task. The two runs are taken from those providing measurements in Figure 5.14. Time 0 indicates the moment when the first workload is submitted to the PanDA system.
as 200 processors, while the run in Figure 5.15(b) utilized only up to 100 processes at maximum. A more distinct difference is that there is always processors available for the run in Figure 5.15(a); in contrast, the run in Figure 5.15(b) cannot be processed continuously due to the lack of available processors. As a result, the total elapsed time of the same task differs largely between the two runs: 9,000 versus 55,000 seconds. We also checked that the total core time consumption of these two runs differ by only 6% (219 versus 232 hours). Thus, the large difference on task elapsed time is mainly due to the resource availability at certain moment in time which can be understood as follows.

As the grid resources have to be shared among data analysis users, the PanDA system schedules user analysis workloads to avoid resources being occupied by a single user. For every fresh user, the first set of workloads will have the highest priority in the system, while the priority decreases gradually as the number of waiting workloads from the same user increases. The prioritized workload is assigned to a site according to its input locality and the estimated turnaround time of the workload. The workloads assigned to a site are processed in order of their priorities.

With this mechanism, the user’s first set of workloads are usually started as soon as the resource provisioning works (i.e. runtime environment setup and user code compilation) are completed. With the decreased priority, the following workloads may not be started immediately since the resources can be taken by the workloads from other users with higher priorities. The competition in getting resources for workloads results in the peaking structure in the evolution of the running workloads over the time.

When the number of high-priority workloads waiting at the site is little, the resources are mostly taken again by the following workloads thus a reasonable number of running workloads can be maintained as shown in Figure 5.15(a). If the site happens to be occupied by high-priority workloads or has a system downtime, the following workloads have to wait for resources to become available again, causing an idle between two batches of workload executions as in Figure 5.15(b) and a longer task elapsed time as the consequence.

Due to this reason, we see that the throughput on the grid is unfortunately rather unpredictable from the user’s point of view since it’s hard to estimate how busy a site will be during the task execution. A possible workaround one can take is to resubmit the waiting workloads to different grid sites; however, this requires additional efforts for workload bookkeeping and management, and the improvement is not guaranteed.

Figure 5.16 shows the evolution of $N_{\text{proc}}$ for a real-life EventLoop task running over the full set of the simulated sample, about 20 times larger than the $t\bar{t}$ sample.

Since the total number of workloads are 20 times larger, they are distributed to more grid sites to utilize more resources. Given a higher number of $N_{\text{proc}}$ is reached at the “initial phase” (i.e. time < 16,000 sec.), the throughput is increased significantly. One
5.3 Data analysis on the Cloud

In the previous section, we see that the data analysis throughput on the grid depends on the evolution of $N_{\text{proc}}$ during the task runtime. The throughput can be high at the beginning; but it decreases due to the competition on resource sharing, resulting in long and unpredictable elapsed time of the task execution. In certain circumstance it can be a serious issue.

In the high-energy physics community, result publication is usually aligned with some particular event such as a conference. It implies that workloads from one data analyzer will encounter a strong competition with those from other analyzers in sharing the grid resources as many of them will be working for the same deadline.

Institutes and data centers usually provide various computing resources (e.g. supercomputer, PC clusters, etc.) to support scientific researches taking place in their own
country. One advantage of those resources is that the system configuration is rather uniform and the management policy is more transparent to their users. However, some local systems may unfortunately have an environment incompatible with the complex of the analysis software, making the adoption of them either hopeless or effortful. Furthermore, those domestic resources are still shared among national users. The competition on resource sharing can still impact on the throughput although it’s not as strong as on the grid.

Cloud computing [138] is a technology emerged recently for delivering computing power “as a service”. One indispensable ingredient of this technology is the resource virtualization by which virtual instances of computing resources (e.g. processors, storages and services) can be created and scaled dynamically in capacity by their users whenever needed. The aim is to provide elastic capacity and the illusion of infinite resources [139]. In addition, certain level of operation and configuration of the virtual resources are under users’ control.

Given these features, the cloud computing has been investigated recently by several data centers operating the WLCG resources. As the resources are virtualized, they can be migrate quickly from one hardware to another. An immediate advantage is the possibility to eliminate service interruption during the hardware maintenance. With the non-interrupted services, data analysis will benefit from continuation of the data processing during the hardware maintenance. Furthermore, infinite resources (even though it’s an illusion) and flexible resource control are attractive features to data analysis users.

For this reason, we tested our data analysis on a cloud system (referred to as the “BigGrid HPC cloud” [140] hereafter) offered by the WLCG Tier-1 center at the Netherlands\textsuperscript{3}. We will demonstrate an approach to integrate virtual resources into the PanDA system and use the virtual resources as extra computing power for data analysis. Furthermore, the throughput of the EventLoop process on this cloud system is also evaluated.

5.3.1 The Infrastructure-as-a-Service (IaaS) cloud system

Generally speaking, computing power can be delivered to user in the form of a computing “machine”, a “platform” providing certain computing capability, or an “application” performing a specific computation. The cloud system that delivers computing power in the form of Virtual Machines (VMs) is called the Infrastructure-as-a-Service (IaaS) cloud system. The BigGrid HPC cloud is one of the IaaS cloud systems provided by the WLCG data centers for scientific computing.

\textsuperscript{3}The infrastructure of the Tier-1 center at the Netherlands is operated by two institutes, naming Nikhef and SARA. The cloud system is hosted at SARA providing on-demand high-performance computing power for scientific researches.
Hardware resources (e.g., CPU, memory, disk) on a computer need to be firstly virtualized so that they can be arranged into virtual machines. Thus, the hardware virtualization forms the basic layer of the IaaS cloud, and it’s achieved by a piece of software called the Virtual Machine Manager (VMM).

The VMs need to be managed in a way to allow rapid and dynamic resource provision within an IaaS cloud system. Being the key component, the Virtual Infrastructure Manager (VIM) interfaces VMM and features dynamic VM placement and management on a pool of physical resources, automatic load balancing, and dynamic infrastructure resizing and partitioning.

At the top of the architecture is the service interface allowing user to interact with the VIM for managing the VMs in the cloud. It can be a web-based interface, a command-line interface or programming APIs.

For the user, resources in a IaaS cloud are acquired by launching VMs. Different to the physical resources on the grid or in a cluster where users are restricted to certain configuration given by the system manager, virtual machines are highly customizable for user applications thanks to the virtualization technology. The customization is achieved by providing a so-called “virtual appliance” which packs application environments such as the operating system and software stacks (e.g., libraries, compilers, databases, etc.). Virtual appliances are commonly packaged as VM disk images which can be directly deployed into the VMM. This approach eases software customization, configuration, and patching.
5.3. Data analysis on the Cloud

Architecture of the BigGrid HPC cloud

The physical resource of the BigGrid HPC cloud is a computer cluster consisting of 19 high-end computing nodes. They share 400 terabytes storage hosted on 2 centralized file servers on which the VM disk images are managed. Each computing node is equipped with 32 CPU cores, 256 gigabytes of memory and 4 network connections of 10-gigabit ethernet. The computing nodes are connected via a centralized network switch providing an end-to-end bandwidth of 40 gigabits between the nodes. The 2 file servers are also connected to the computer nodes via the same switch sharing a bandwidth of 40 gigabits.

On top of the physical resource layer, the BigGrid HPC cloud uses KVM [141] for hardware virtualization and OpenNebula [142] for virtual infrastructure management. User can deploy, manager and monitor VMs in the cloud through a self-service web portal, or using a programing interface based on the XML-RPC protocol.

Integration with the PanDA system

To make the adoption of the cloud resource transparent to our data analysis workflow, we integrate the virtual resources into the PanDA system so that the developments we have discussed before can be fully reused.

With the pilot framework, the integration is simply done by presenting the BigGrid HPC cloud to the PanDA system as a “virtual” grid site. Workloads to be processed by the BigGrid HPC cloud are then assigned by the PanDA system to this virtual grid site, waiting for VMs to be started in the cloud and configured to run the pilot jobs.

The Ganga toolkit is extended to interface with the BigGrid cloud, providing a faster and scalable way of launching, monitoring and shutting down VMs. Once the VMs are launched in the cloud, they are configured to form a computer cluster running the Condor batch system. Pilot jobs are then submitted constantly to the cluster to run on the virtual machines, and the workload execution through pilot job runs as it does on the grid.

5.3.2 Throughput of the EventLoop process

We evaluate the EventLoop throughput on the cloud using the same task defined before for measuring the throughput on the grid. Figure 5.18 shows the distributions of the CPU and wall-clock time of the workloads running on the cloud. Comparing to the workloads on the grid (Figure 5.12), both the CPU and wall-clock time have relatively smaller deviations on the cloud. The reason has two folds. Firstly the resources on the cloud is completely uniform in terms of hardware specification. Secondly the resources are also customized uniformly from the operation system to the application.
Chapter 5. Optimizing Throughput of User Data Analysis

We also notice from the Figure 5.18 that the average CPU time consumption on the cloud is close to it on the grid (Figure 5.12), suggesting that the cost of CPU virtualization is negligible to the EventLoop process.

However, the wall-clock time consumption is the main concern for the throughput; and on the cloud, the value (701 sec.) is larger than the average on the grid (569 sec.). This implies a lower CPU utilization fraction. Figure 5.19 adds the CPU utilization

![Figure 5.18: Distributions of the CPU and wall-clock time of the EventLoop workloads running over 10,000 events on the cloud. The mean values and the standard deviations are shown in the legend.](image)

![Figure 5.19: Comparison of the CPU utilization fractions between workloads on the grid sites and on the BigGrid HPC cloud. Grid sites are indexed by alphabets, while the BigGrid HPC cloud is indicated by “CL”.](image)
fraction for the cloud to the plot of Figure 5.13. It shows that the fraction for the cloud (site indexed with “CL”) falls actually into the lower side and it is just about 50%.

Given that the virtual machine’s local disk is actually a file (i.e. disk image) stored on the centralized file server, the workloads on the cloud are configured to read events over the network, for avoiding disk I/O on the virtual machine. As Figure 5.6 suggests that the I/O protocol has influence on the event reading performance, we investigate further its impact on the throughput. From Figure 5.20, we observe that the CPU utilization fraction can be increased by utilizing a different I/O protocol for event reading.

![Figure 5.20: Comparison of the CPU utilization fractions between workloads using different I/O protocols for event reading. The point of “gsidcap@SARA” corresponds to the point of “CL” in Figure 5.19.](image)

The comparisons on $T_{core}$ and $T_{elapsed}$ for different I/O protocols are also shown in Table 5.1. Unlike our observation on the grid, we notice here that $T_{elapsed}$ reduces as the increase of the CPU utilization fraction, suggesting that the throughput of the cloud can be enhanced effectively by increasing the CPU utilization fraction. It can be done by, for instance, switching to a different I/O protocol. We also notice that the throughput on the cloud system can be tuned in this way to have higher throughput than the average on the grid. But one should keep in mind that the numbers presented on the table are measured with a task representing only 1/20 of the full data analysis.

Table 5.1 also shows the expected correlation between $T_{core}$ and $T_{elapsed}$ given the fact that $N_{proc}$ for the cloud is rather a constant and under users’ control.

In Figure 5.21, the time evolution of the number of running workloads shows that the $N_{proc}$ has a ramping-up phase at the beginning due to the overhead of the virtual
Table 5.1: Total core time consumption and the elapsed time of the EventLoop task running on the BigGrid HPC cloud with different I/O protocols, comparing to their means of 20 runs on the grid. The throughput is calculated as Equation 5.2 where $N_{events}$ is $1.626 \times 10^7$. The EventLoop task is performed on the cloud with 20 virtual machines.

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Core Time ($T_{core}$)</th>
<th>Elapsed Time ($T_{elapsed}$)</th>
<th>Throughput (events/second)</th>
</tr>
</thead>
<tbody>
<tr>
<td>gsidcap@SARA</td>
<td>310</td>
<td>16</td>
<td>282</td>
</tr>
<tr>
<td>rpio@NIKHEF</td>
<td>228</td>
<td>12</td>
<td>376</td>
</tr>
<tr>
<td>xrootd@SARA</td>
<td>181</td>
<td>9</td>
<td>502</td>
</tr>
<tr>
<td>Grid (mean)</td>
<td>242</td>
<td>12</td>
<td>369</td>
</tr>
</tbody>
</table>

Machine deployment. Once the resources are occupied, they remain available for the entire task thus a constant value of $N_{proc}$ is maintained. With this feature, one can increase $N_{proc}$ by deploying more virtual machines in the cloud and expect the total elapsed time to be shortened accordingly.

![Figure 5.21](image)

**Figure 5.21:** Time evolutions of the number of running workloads and the accumulated number of finished jobs from two runs of the same EventLoop task on the cloud, using (a) 20 and (b) 40 virtual machines. For both runs, the same I/O protocol (gsidcap@SARA) is used. Time 0 indicates the moment when the first workload is submitted to the PanDA system.

Although the comparison between Figure 5.21(a) and Figure 5.21(b) shows a reduction of the elapsed time by roughly a factor of 2 after doubling the number of processors,
our question here is how the throughput is actually improved in accordance with the increase of $N_{\text{proc}}$. To answer the question, we measure the “speedup” defined as

$$\text{speedup} = \frac{T_{\text{elapsed}}(N_{\text{proc}} = 10)}{T_{\text{elapsed}}(N_{\text{proc}} = n)} ; n = 20, 40, 80$$

(5.5)

Here the speedup is defined with respect to $N_{\text{proc}} = 10$ instead of $N_{\text{proc}} = 1$ since it is simply take too long (320 hours in estimation) for one virtual machine to complete the task.

From Figure 5.22, we observe that the speedup increases linearly up to 40 processors, which implies that the demonstrated throughput for the cloud in Table 5.1 continues to scale by a factor of 2. However, a significant deviation from linear occurs with 80 processors, indicating that the cloud system is not able to cope with the scale required for running over the full set of data.

In Figure 5.16, we see that running over full data set on the grid takes roughly 18 hours of elapsed time. According to Table 5.1, the smallest $T_{\text{elapsed}}$ achieved on the cloud for our test task running on 1/20 of the full data set is 9 hours using 20 VMs, which can be reduced linearly to 4.5 hours with 40 VMs. With the guaranteed resource availability, the needed $T_{\text{elapsed}}$ on the cloud for analyzing a full data set is estimated to be $20 \times 4.5 = 90$ hours, 5 times longer than it takes on the grid.

Further investigation reveals that the bottleneck is on the central file server where all the virtual machine images are stored. One should note that the virtual machine image serves as a local disk, every disk I/O from the operation system and applications generates load to the file server. When the load exceeds the file server’s capability, all the virtual machines in the cloud will be slowed down.
5.3.3 Outlook

Since the cloud computing is a relatively new technology to the LHC computing, this study investigates only a small aspect of the cloud system. Although the guaranteed resource availability is beneficial to mission critical analysis, there are still issues to be addressed in the future.

Given the typical setup as the BigGrid HPC cloud system, we have observed a scalability issue due to the centralized file system. One workaround to avoid the scalability bottleneck is to configure the IaaS cloud system in such that the virtual machine image is copied to the disk of the physical machine on which the virtual machine runs. However, it introduces an extra overhead of virtual machine deployment and sacrifices the live migration feature\(^4\) of the IaaS cloud system. Another possible solution would be hosting virtual machine images on a distributed file system, by which the load from intensive disk I/O on virtual machines is distributed across file servers. With this solution, hardware cost and maintenance effort are two main concerns.

In this study, we assume that data is close to the cloud system so that the event reading over network utilizes only the Local Area Network (LAN). Given the fact that data is distributed around the globe, how to arrange the access to remote datasets for best event reading performance is still an open question. We see several developments nowadays with potential to address this question in the future.

There is an on-going development of the EGI Federated Cloud \([143]\) which integrates the cloud systems offered by the WLCG data centers for international scientific collaborations. According to its blueprint \([144]\), user will be able to launch virtual machines in a remote cloud system that is relatively close to data.

With the improved connectivity between grid sites, another possibility is to perform event reading right over the Wide Area Network (WAN). On this subject, efficient directory service for data lookup and advanced data caching mechanism are required for reliability and performance. As of today, two approaches are being developed within the WLCG community. Although the common goal is to establish a storage federation in which users are able to access distributed data using one single namespace, the two approaches adopt different I/O protocols. FAX \([145]\) choses the XROOTD protocol which is natively supported by the ROOT framework and well optimized for the HEP applications; while another development \([146]\) choses the HTTP/WebDAV protocol which is nowadays widely supported by various grid and cloud storage technologies. It would be interesting to see how the user analysis throughput will be affected when data access over WAN is enabled within the storage federations.

\(^4\)The live migration feature refers to the process of moving a virtual machine from one physical hardware to another without disconnecting the clients and the applications running on it.
5.4 Summary

In the search for single top quark production in the $Wt$-channel, the data analysis workflow is decomposed into steps where the most time consuming computations are involved in the EventLoop process. Given its direct impact on the performance of our data analysis and the quality of physics measurements, we evaluated and optimized the throughput of the EventLoop process using the grid and the cloud computing technologies.

For analyzing a large amount of data, the needed EventLoop throughput can only be achieved by distributing workloads widely on the grid. However, due to competitions in resource sharing, the amount of resources available for user analysis can vary largely from time to time during the life cycle of the user analysis, causing the throughput on the grid to be largely uncertain.

We showed how a more predictable throughput can be achieved by using an IaaS cloud system. With more control on resource allocation, we observed that the throughput on the cloud can be increased by either adopting a different I/O protocol for event reading or employing more virtual machines. On the other hand, we also observed that the centralized file system enabling the live migration feature of the IaaS cloud can become a scalability bottleneck when the file system experiences high load from intensive disk I/O on the virtual machines. For our analysis, the cloud is missing the scalability by a factor of 5 to truly compete with the grid due to the bottleneck. This bottleneck as well as the question about how to access efficiently a remote datasets from the cloud need to be addressed in the future for exploiting the full capacity of the cloud system. The challenge will be to achieve the scalability while retaining the advantages of the cloud.