Published in final edited form as:
Proc SPIE Int Soc Opt Eng. 2016 February 27; 9789: . doi:10.1117/12.2217396.

Performance Management of High Performance Computing for Medical Image Processing in Amazon Web Services

Shunxing Bao[a], **Stephen M. Damon**[b], **Bennett A. Landman**[a,b], and **Aniruddha Gokhale**[a,b]

[a]Computer Science, Vanderbilt University, Nashville, TN, USA 37235

[b]Electrical Engineering, Vanderbilt University, Nashville, TN, USA 37235

Abstract

Adopting high performance cloud computing for medical image processing is a popular trend given the pressing needs of large studies. Amazon Web Services (AWS) provide reliable, on-demand, and inexpensive cloud computing services. Our research objective is to implement an affordable, scalable and easy-to-use AWS framework for the Java Image Science Toolkit (JIST). JIST is a plugin for Medical-Image Processing, Analysis, and Visualization (MIPAV) that provides a graphical pipeline implementation allowing users to quickly test and develop pipelines. JIST is DRMAA-compliant allowing it to run on portable batch system grids. However, as new processing methods are implemented and developed, memory may often be a bottleneck for not only lab computers, but also possibly some local grids. Integrating JIST with the AWS cloud alleviates these possible restrictions and does not require users to have deep knowledge of programming in Java. Workflow definition/management and cloud configurations are two key challenges in this research. Using a simple unified control panel, users have the ability to set the numbers of nodes and select from a variety of pre-configured AWS EC2 nodes with different numbers of processors and memory storage. Intuitively, we configured Amazon S3 storage to be mounted by pay-for-use Amazon EC2 instances. Hence, S3 storage is recognized as a shared cloud resource. The Amazon EC2 instances provide pre-installs of all necessary packages to run JIST. This work presents an implementation that facilitates the integration of JIST with AWS. We describe the theoretical cost/benefit formulae to decide between local serial execution versus cloud computing and apply this analysis to an empirical diffusion tensor imaging pipeline.

Keywords

Cloud deployment; Amazon Web Service; High performance computing; NITRC-CE

1. Introduction

The increasing accumulation of open-source multi-modal images makes it imperative to deploy local processing of images into cloud-based environments. An example of such a image processing framework is the Java Image Science Toolkit (JIST) [1–3], a biomedical plugin for MIPAV[4], which is entirely developed in the Java programming language[5,6] and is hosted on the Neuroimaging Informatics Tools and Resource Clearinghouse (NITRC). Amazon Web Services (AWS) provide reliable, on-demand, and affordable cloud computing services [7] that is promising for medical image processing. Different types of optimized

Amazon Elastic Cloud Computing (EC2) instances are offered in AWS [8]. Amazon Simple Storage is easy to apply, cost oriented, and highly scalable as data/object storage [9]. Despite the promise, workflow definition/management and cloud configurations are two key challenges in this realm. We exploited the NITRC Computational Environment (NITRC-CE) [10,11] hosted by Amazon's AWS, to distribute each JIST "Execution Context" (process) on a separate cloud node. Through an applicable cost/benefit analysis, users are able to specify a grid size, automatically boot up a private network, install JIST into MIPAV, and set up all requirements for processing. All monitoring is managed through the standard JIST Process Manager.

Our contributions are summarized as follows.

- We provide a concrete workflow definition/management for JIST to deploy a local project to be distributed and processed in the cloud.
- We integrate JIST to Amazon AWS (a reliable, on-demand, and inexpensive cloud computing services) to execute high performance computing.
- We demonstrate a cost/benefit analysis of a pipeline that can be easily parallelized and requires memory that would limit the processes to serially execute only on a local lab machine.

This paper is organized as follows. In Section 2, we introduce in detail the JIST cloud deployment framework with a glimpse of its implementation. We present cost/benefit analysis especially focusing on project total running time estimation. Section 3 describes performance analysis for a multi-subject diffusion tensor imaging experiment.

2. Method

2.1 Workflow framework

Cloud configuration is based on Amazon AWS (of which S3 storage and EC2 are two components). Amazon S3 storage is mounted by every pay-for-use Amazon EC2 instance. Hence S3 is recognized as a shared cloud resource for all EC2 machines that are ordered by clients. S3 stores all necessary packages including MIPAV's install file, JIST plugins, JIST Library and other files related to configuration. Each Amazon EC2 machine can get the source packages from S3, and install related packages to run JIST.

Briefly, JIST pipelines are stored in a "layout" file, which specifies inputs and outputs as well as algorithm classes used for execution. The user's layout is later sent to S3 storage, and all processing is handled in the cloud virtual machines. The local host (local machine) meanwhile monitors the processing process in cloud. Rather than run whole layout on a typical cloud machine, each processing command (a JIST "execution context", which is a single process in the process manager) generated by the JIST scheduler is distributed to any of the free EC2 instances. The process manager periodically sends jobs that are ready to run to the cloud when idle machines are found. Each job has a local monitor that periodically pulls back its debug and error info in the cloud so that users can view the progress of a running job. Status "Complete" represents a process that has completed, and all files that have been produced during the proceeding have been replicated back to local workstation.

Note, to copy files between systems, the directory paths must be modified to represent differences in mount points and local structures. Thus, outputs are available to operate on locally even though processing happens on AWS. The workflow framework is presented in Figure 1.

2.2 Configuration

Several modes to run a layout can be selected in the JIST layout preference panel (e.g., run on a DRMAA compliant grid, run locally, or run in the Amazon Cloud). Users should choose "use AWS EC2" option to activate cloud computing (Figure 2-(a)).

Using the Cloud Dashboard, users are able to configure, initialize ("init"), boot, and terminate an Amazon AWS NITRC-CE cloud instance with predefined machine information (Figure 2(b)). Access Key Id, Secret access key and region information are used to validate an AWS account so that user can access Amazon EC2 and S3 resources. Custom S3 buckets are a top-level folder on S3, the name of which must be globally unique in Amazon S3. This bucket is used to store necessary packages and JIST layout projects with the same structure. A standard cryptographic key pair is needed to securely log in and connect to remote cloud instances. Inbound and outbound connection rules including IP and port permission range are defined in security groups. Also, users can select several different types of pre-configured Amazon machines and any number of identical machines to boot. These machines have different processor speeds and memory installments. The initialize and install processes are executed once all necessary files and settings are prepared. In JIST's "Progress Manager", project input files are then uploaded to cloud and they wait for processing.

2.3 Cost/Benefit analysis

Time usage and expense are two integral factors to be considered cloud computing. Here, we consider if there is a way to estimate cloud usage costs relative to running serially on local lab machines. Total runtime in the cloud can be estimated by the following where $\boldsymbol{T_A}$ represents runtime in the cloud.

$$T_A = f(A, B, S_{\text{data}}, S_{\text{out}}, g(n)) = \frac{1}{A}\left(\frac{S_{\text{data}} + S_{\text{out}}}{B} + g(n)\right)$$

for the Amazon machine instance type $\boldsymbol{A}$. The above variables are defined in detail in the following sections. Briefly, $\boldsymbol{B}$ is bandwidth, $\boldsymbol{S_{data}}$ is input data size, $\boldsymbol{S_{out}}$ is the output data size, $\boldsymbol{g(n)}$ is the function to compute estimated execution time for all jobs, and $\boldsymbol{n}$ is the total number of cloud instances.

AWS provides a considerable spectrum of instances family for on-demand purpose, i.e., "General Purpose" (balance of compute, memory, and network resources), "Compute Optimized" (highest performing processors with lowest cost/compute), "Memory Optimized" (memory-intensive applications and have the lowest cost/GiB of RAM), and "Storage Optimized" (SSD storage with low cost high IOPS) [9]. Different types of instances have different processors, which need to be considered when predicting performance. Within each instance class, instances use same kind of processors with different total number. Memory should also be considered into install process (especially means installing MIPAV in each EC2 machines) and each job-processing time estimation. $\boldsymbol{A}$ is a weight value between (0, 1), when smaller cache and low memory allocation instance is applied, the value of A is more close to 0 (i.e., the machine is slower).

Input data size $\boldsymbol{S_{data}}$: Input image data size is also used to estimate processing time of a single job. In the example that we present, the process nearly linearly increases as the number of voxels increase. Amazon machine instance type $\boldsymbol{A}$ also affects the job time when taking processor cache and memory into consideration.

Bandwidth $\boldsymbol{B}$: When running a big layout with lots of data that gets generated and multiple processing algorithms, upload/download time is not a primary factor in the model. However, if a layout is small, it is worth considering as the transfer process may end up taking as long, if not longer, than the processing time. On the same note, installation processes are also taken into consideration. The NITRC-CE includes many popular image processing toolkits, but is not guaranteed to have a compatible version of MIPAV and JIST --- these are transmitted and installed for each newly created cloud node in the S3 bucket. In general, a small layout is a project when it is applied only one single process to run the entire job, the total time is smaller than an hour, and the process needs less than 500G memory. The uploading time depends on the size of the project, or $\boldsymbol{S_{data}}$, because the file size is usually bigger than any other type of input files in the JIST project (i.e., ".input", ".output", ".LayoutXML"). However, the output file size(s) $\boldsymbol{S_{out}}$ need to be estimated. Subsequently, install upload and download time is then computable via bandwidth $\boldsymbol{B}$.

Before explaining the details of $\boldsymbol{g(n)}$, we first introduce the job dependency tree $\boldsymbol{D}$. JIST's scheduler can handle the graph of execution dependencies. A lower priority value results in later execution and vice versa. The scheduler itself can also estimate execution time of a job. If several jobs are ready to run with the same priority at the same time, the job that has the smallest execution time will run first. For example, there are 4 jobs are all ready to run with same priority value, the estimation execution time is: job-a (15 minutes), job-b (18 minutes), job-c (50 minutes) and job-d (60 minutes). If two free instances are available, job-a and job-b will start first, and this will cost less time than any other combination of 2 jobs to run first. As a result, a job dependency graph can be generated. This can help a user to decide the total number of instances.

$\boldsymbol{D}$ is a depth-first-search tree, according to job dependency. We first build an entire tree, and denote each node as slow and fast to represent processing time of a job. Then we remove all fast nodes, and connect all slow nodes via the hierarchy. Figure-3 presents an example of how $\boldsymbol{D}$ is generated.

Based on $\boldsymbol{D}$, we first calculate the total number of nodes on each level and get the maximum number among all levels. We can then get an estimated number of maximum processes that can run at once. The average number of machines can be derived from a level in $\boldsymbol{D}$**,** which has the maximum total execution time and maximum single running time in the selected level. If $\boldsymbol{T_{i|i=1,2,3\ldots m}}$ is processing time of m jobs in a chosen level, average number of instances then can be calculate by:

$$\mathrm{avg}(n)=\left[\frac{\sum_{i=1}^{m} T_i}{\mathrm{MAX}(T_{i|i=1,2..m})}\right]$$

More machines can be booted with the benefit of decreased execution time at the expense of increased cost.

Once ***n*** is set, if there are ***x*** paths in ***D***, we group ***x*** paths into ***y*** groups, where y equals ***x/n***, and $\boldsymbol{T_y}$ is max time to run all jobs in group ***y***. $\boldsymbol{T_{duplicate}}$ is the duplicate time among each groups. Finally, the estimation time to process all jobs in the cloud is

$$g(n)=\sum_{i=1}^{y} T_i - T_{\text{duplicate}}$$

According to $\boldsymbol{T_A}$, type of instance and the number of nodes, the expense ***E*** is easy to calculate as

$$E=|T_A| \cdot \mathrm{num(Inst)} \cdot \mathrm{price(A)}$$

A small amount of data transfer and data storage fee also occurs, but it is considerably less than the cost of renting EC2 machine.

The tradeoff between number of instances booted and cost is one of the most important considerations to make. For example, consider a layout where there are many fast jobs that can run simultaneously, and they are all depending on a single slow job. In this scenario, we should cautiously select the number of machines to reduce the idle time of instances, as this is payment for non-processing time.

Different types and numbers of instances lead to different T_A, which incurs several pairs of (E,T_A). Based on budget and expectant time, users can decide if JIST cloud mode is applicable.

3. Case study and Results

Our test experiment is a multi-subject diffusion tensor imaging (DTI) project [10]. DTI is a method to non-invasively measure water diffusion in the human brain using magnetic resonance imaging (MRI). In this example, three different subjects are in the layout. The layout is used to fit tensors to the acquired data using four different tensor fitting routines (Figure-4). The input data size is 256*256*65*D, where D is the number of directions at which different gradients are applied and readout. In this case, there are 34. Table-2 illustrates the experiment top-down structure. Level means the job hierarchy and dependency.

The final Job dependency DFS tree ***D*** is shown in Figure-5. In summary, the total number of level-2 in ***D*** is larger than level-1, so the maximum number of processes that are running at the same time in our experiment is 12. Moreover, they are all slow jobs, so we should consider applying multiple instances rather than a single instance. In our experiments, in order to control variable, we apply the same Amazon instance family (namely M3), the processors of which is High Frequency Intel Xeon E5-2670 v2 (Ivy Bridge) Processors.

3.1 Case 1: Same total number of instance *n*, different Amazon instance type *A*

As a brute force approach, for each instance type, we all apply maximum number of instances 12. Table-3 presents the result of actual total running time of each option with total cpu time in cloud and expense. Total Cpu time in cloud is the time using only a single process to execute entire jobs in one virtual machine, which is also regarded as run layout in local workstation. Figure 6 visually shows the performance of JIST cloud mode compared with total cpu time.

The result shows the estimation time $\boldsymbol{T_A}$ is reasonable with 90% precision. It illustrates that Medium instances can accomplish processing the project for a low cost (a bit more than Large instances'), but approximately with double the needed time over other instances type because of lack of memory allocation. The result also demonstrates ×Large's memory is big enough for our experiment with a total experiment time of 4 hrs 10 mins. The time does not dramatically decrease even when we apply a more expensive type instance (e.g., 2×Large). However, the total expense of 2×Large is much more than the other 3 instance types. Note that the expense of Medium is even more than Large instances. Running 12 jobs at the same time is hard for a normal local station, while running 12 jobs with 12 available instances is easy to accomplish. Figure 7 depicts these 12 jobs running all at the same time.

3.2 Case 2: Same instance type (large), different number of machines *n*

Due to job top-down structure, for one subject, 4 tensor fitting methods can run simultaneously. We use it to compute ***avg(n)*** equals 1.68 which means an average of 2 instances can be applied for a subject. Thus, we should apply 6 instances in total for entire project. Table 4 demonstrates that 12 large machines just save 16 minutes (4%) more than 6 large machines, which is because of the structure of the project.

Before tensor fitting, at most 6 machines will be used, so basically 6 machines will be idled when we apply 12 machines. While when we perform tensor fitting, 12 machines can save partially parallel time. When using 12 machines, 12 tensor fitting algorithm are run at the same time. While when 6 machines are ordered, owing to JIST scheduler, CAMINO-WL and CAMINO-LLMSE are completed first, and job CAMINO-NL and Single tensor Estimation are then run. In this case, total number of 6 large instances should be ordered since it saves 40% expense and just need 16 more minutes to complete the entire jobs.

4. Conclusion

This paper has presented a system to enable automatic deployment of a JIST to the Amazon AWS cloud. Each processes of user's project are processed in separate nodes in Amazon AWS. Mounting Amazon S3 storage to all EC2 is automatically accomplished for efficient I/O and data transfer in cloud environment. Our cost/benefit analysis can help users to decide grid size and configuration relative to local computation. A performance prediction scheme to accurately predict computing resource is found in [14], stricter cost estimation could be applied to current work. EC2 limit performance compared with conventional HPC platform has also introduced in [11]. So our future work is to maintain JIST cloud version and exploit cloud-based JIST to run very large parallel processes with efficient use of EC2 machines.

Acknowledgments

This work is supported by R01EB15611 and NIH 1R03EB012461. The project described was supported by the National Center for Research Resources, Grant UL1 RR024975-01, and is now at the National Center for Advancing Translational Sciences, Grant 2 UL1 TR000445-06. The content is solely the responsibility of the authors and does not necessarily represent the official views of the NIH.

REFERENCES

1. Li B, Bryan F, Landman BA. Next Generation of the Java Image Science Toolkit (JIST): Visualization and Validation. The insight journal, 2012. 2012:1–16.Davis AR, Bush C, Harvey JC, Foley MF. Fresnel lenses in rear projection displays. SID Int. Symp. Digest Tech. Papers. 2001; 32(1):934–937.
2. Lucas B, Landman B, Prince J, Pham D. MAPS: a free medical image processing pipeline. Organization for Human Brain Mapping. 2008
3. Lucas BC, Bogovic JA, Carass A, Bazin PL, Prince JL, Pham DL, Landman BA. The Java Image Science Toolkit (JIST) for rapid prototyping and publishing of neuroimaging software. Neuroinformatics. 2010; 8(1):5–17. [PubMed: 20077162]
4. McAuliffe MJ, Lalonde FM, McGarry D, Gandler W, Csaky K, Trus BL. Medical image processing, analysis and visualization in clinical research. 2001:381–386.
5. Boisvert RF, Moreira J, Philippsen M, Pozo R. Java and numerical computing. Computing in Science & Engineering. 2001; 3(2):18–24.
6. Bull JM, Smith LA, Pottage L, Freeman R. Benchmarking Java against C and Fortran for scientific applications. 2001:97–105.
7. Amazon Web Services (AWS) – Cloud Computing Services [Online]: http://aws.amazon.com.
8. Amazon Elastic Computing – Scalable Cloud Hosting [Online]: http://aws.amazon.com/ec2/.
9. Amazon Simple Storage Service (S3) [Online]: https://aws.amazon.com/s3/.
10. Luo XZ, Kennedy DN, Cohen Z. Neuroimaging informatics tools and resources clearinghouse (NITRC) resource announcement. Neuroinformatics. 2009; 7:55–56. [PubMed: 19184562]
11. Kennedy, DN.; Haselgrove, C. The three NITRC's: software, data and cloud computing for brain science and cancer imaging research; Front Neuroinform; Conference Abstract: Neuroinformatics; 27 Aug-29 Aug, 2013; Stockholm, Sweden. 2013.
12. Amazon EC2 Instance Types[Online]. Available: http://aws.amazon.com/ec2/instance-types/.
13. Landman B, Huang A, Gifford A, Vikram D, Lim I, Farrell J, Bogovic J, Hua J, Chen M, Jarso S, Smith S, Joel S, Mori S, Pekar J, Barker P, Prince J, van Zijl P. Multi-Parametric Neuroimaging Reproducibility: A 3T Resource Study. Neuroimage. 2011; 4:2854–2866. [PubMed: 21094686]
14. Zhang HL, Li PP, Zhou ZG, Du XJ, Zhang WZ. A Performance Prediction Scheme for Computation-Intensive Applications on Cloud. Proc. of ICC 2013. 2013
15. Jackson KR, Ramakrishnan L, Muriki K, Canon S, Cholia S, Shalf J, Wasserman HJ, Wright NJ. A Performance Prediction Scheme for Computation-Intensive Applications on Cloud. Proc. Of CloudCum 2010. 2010

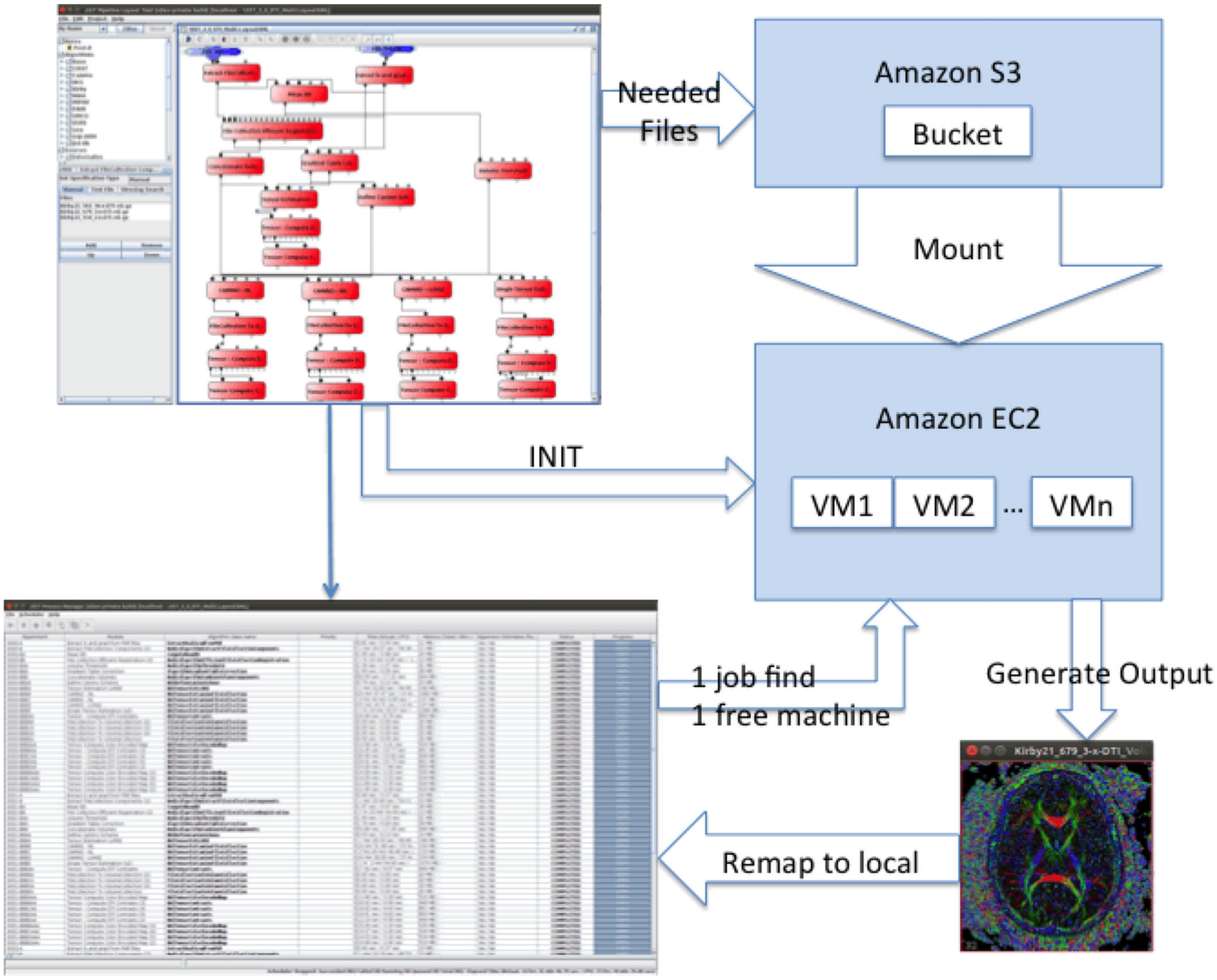
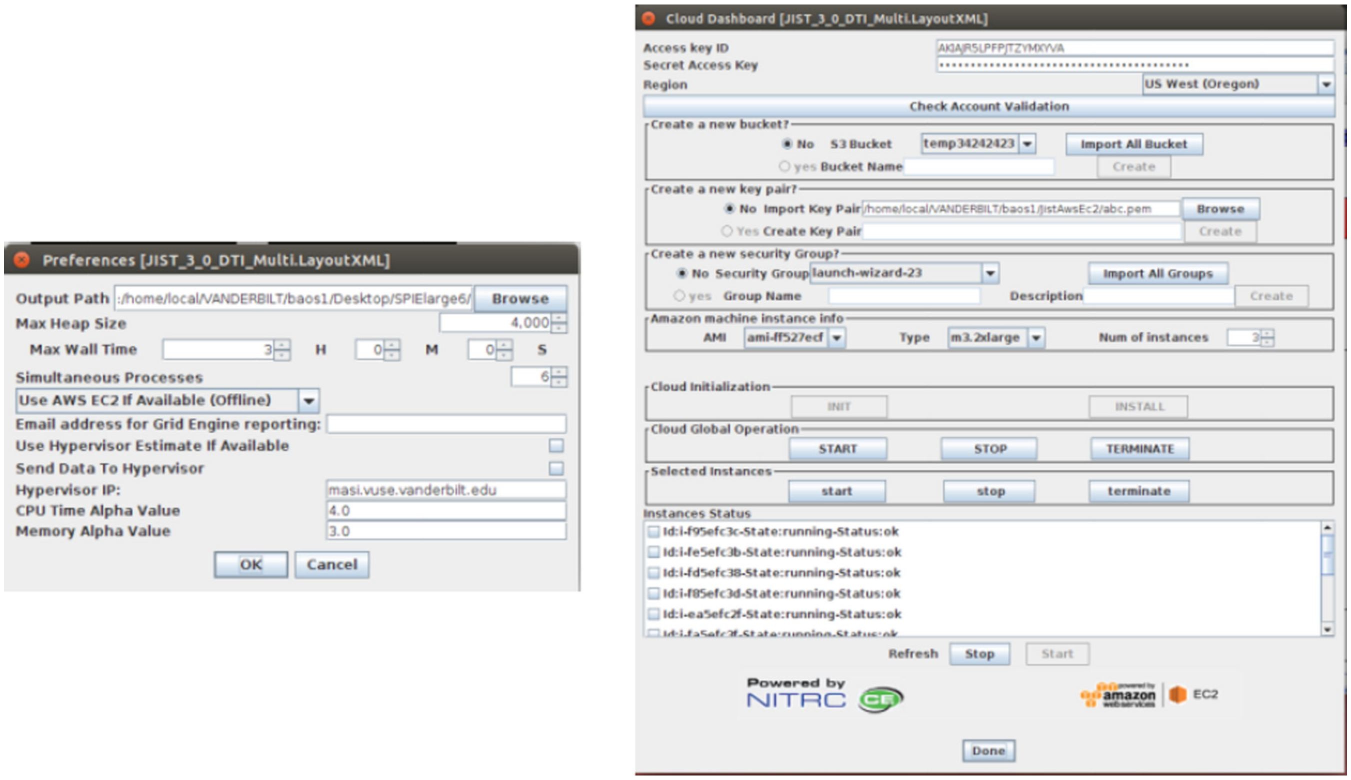


Figure 1.
Workflow framework.


Preferences [JIST_3_0_DTI_Multi.LayoutXML]

Output Path :/home/local/VANDERBILT/baos1/Desktop/SPIElarge6/ Browse
Max Heap Size 4,000
Max Wall Time 3 H 0 M 0 S
Simultaneous Processes 6
Use AWS EC2 If Available (Offline)
Email address for Grid Engine reporting:
Use Hypervisor Estimate If Available
Send Data To Hypervisor
Hypervisor IP: masi.vuse.vanderbilt.edu
CPU Time Alpha Value 4.0
Memory Alpha Value 3.0
OK Cancel

Cloud Dashboard [JIST_3_0_DTI_Multi.LayoutXML]

Access key ID AKIAJR5LPFPJTZYMXYVA
Secret Access Key ••
Region US West (Oregon)
Check Account Validation
Create a new bucket? [✓ No] S3 Bucket temp34242423 Import All Bucket
yes Bucket Name Create
Create a new key pair? [✓ No] Import Key Pair /home/local/VANDERBILT/baos1/jistAwsEc2/abc.pem Browse
Yes Create Key Pair Create
Create a new security Group? [✓ No] Security Group launch-wizard-23 Import All Groups
yes Group Name Description Create
Amazon machine instance info AMI ami-ff527ecf Type m3.2xlarge Num of instances 3
Cloud Initialization INIT INSTALL
Cloud Global Operation START STOP TERMINATE
Selected Instances start stop terminate
Instances Status
Id:i-f95efc3c-State:running-Status:ok
Id:i-fe5efc3b-State:running-Status:ok
Id:i-fd5efc38-State:running-Status:ok
Id:i-f85efc3d-State:running-Status:ok
Id:i-ea5efc2f-State:running-Status:ok
Id:i-fa5efc3f-State:running-Status:ok
Refresh Stop Start
Powered by NITRC CE
powered by amazon web services EC2
Done

Figure 2.
(a) JIST preferences panel (b) JIST Cloud Dashboard

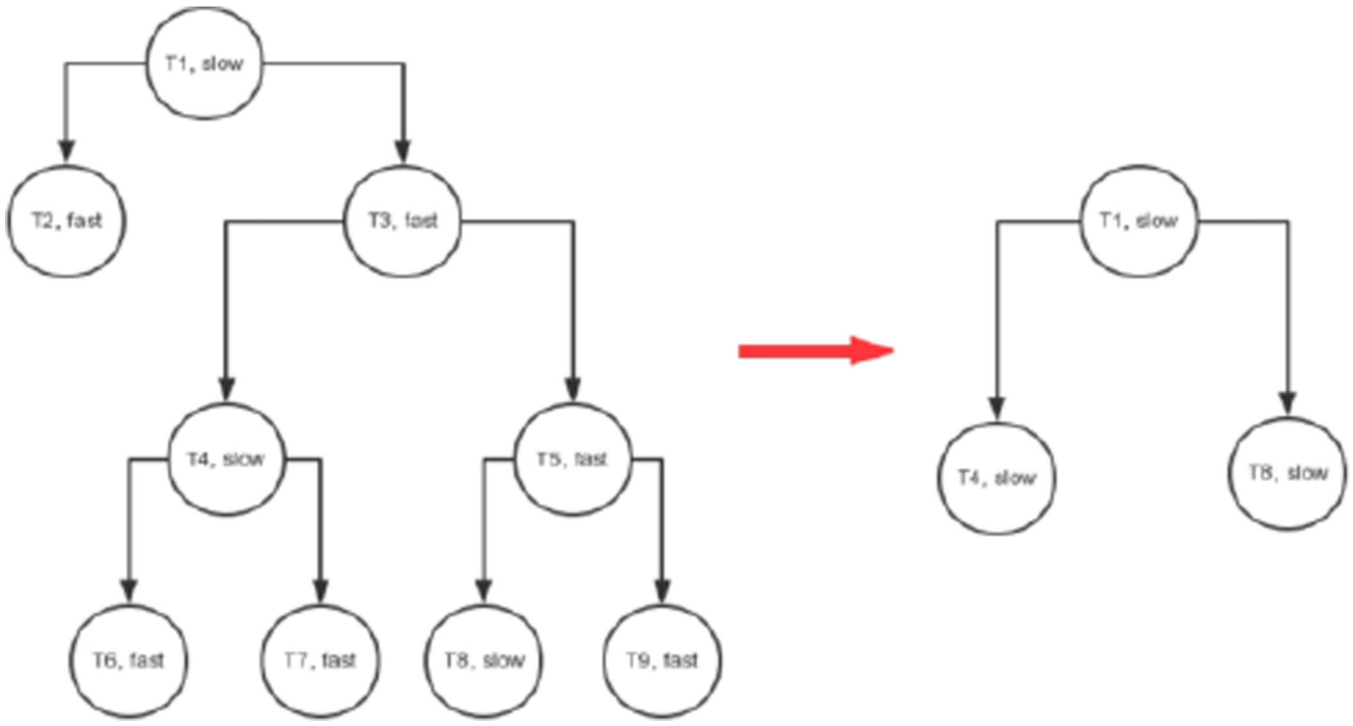
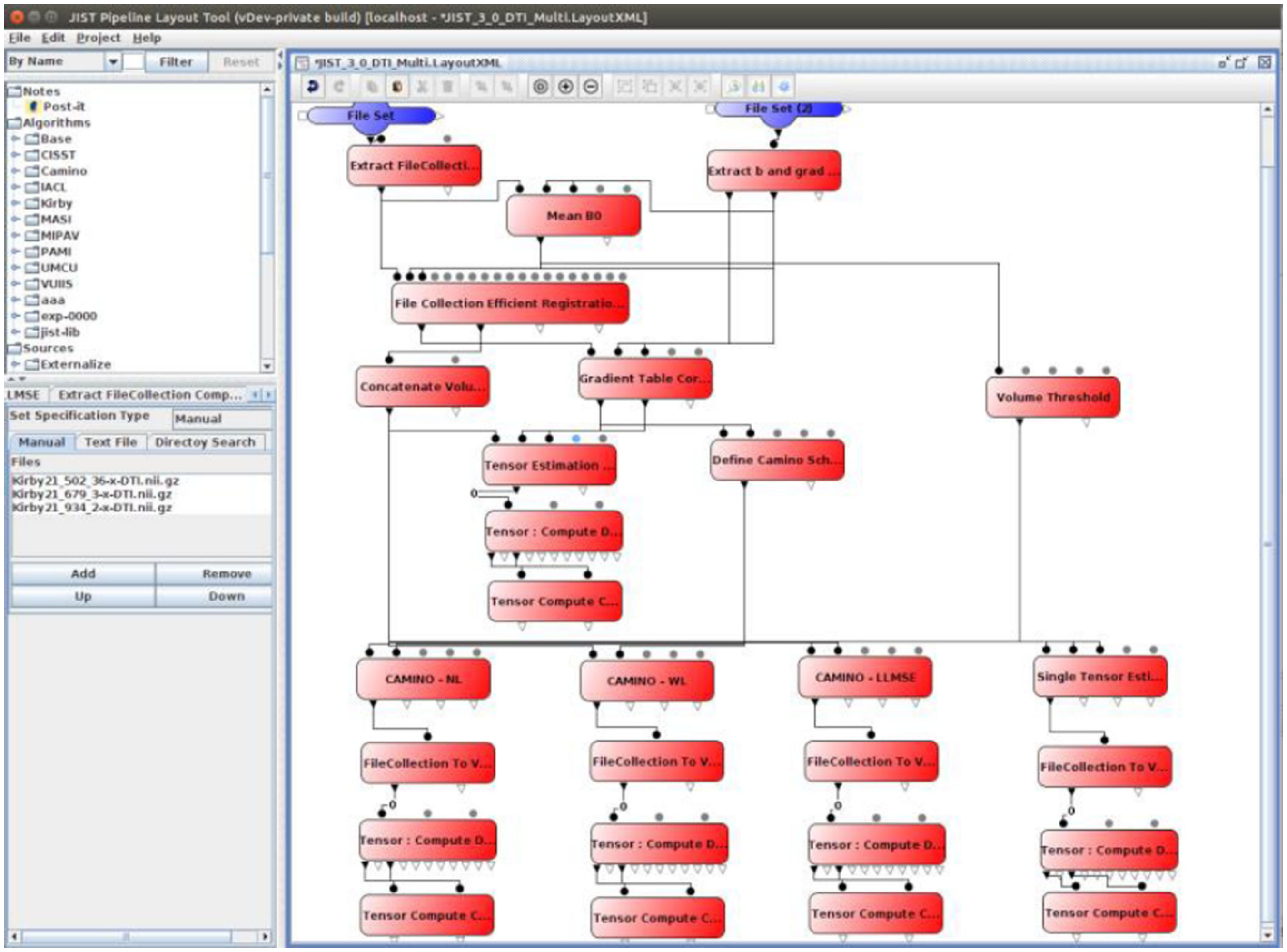


Figure 3.
Job dependency DFS tree generation



JIST Pipeline Layout Tool (vDev-private build) [localhost - *JIST_3_0_DTI_Multi.LayoutXML]

Set Specification Type: Manual

Files

Kirby21_502_36-x-DTI.nii.gz
Kirby21_679_3-x-DTI.nii.gz
Kirby21_934_2-x-DTI.nii.gz

Figure 4.

Multi-slice DTI test layout.

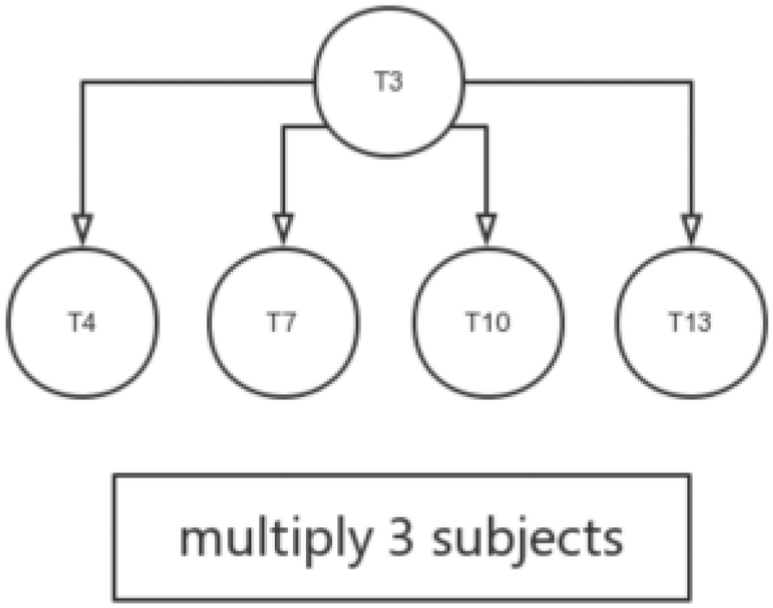


Figure 5.
Job dependency DFS tree for multi-slice DTI experiment

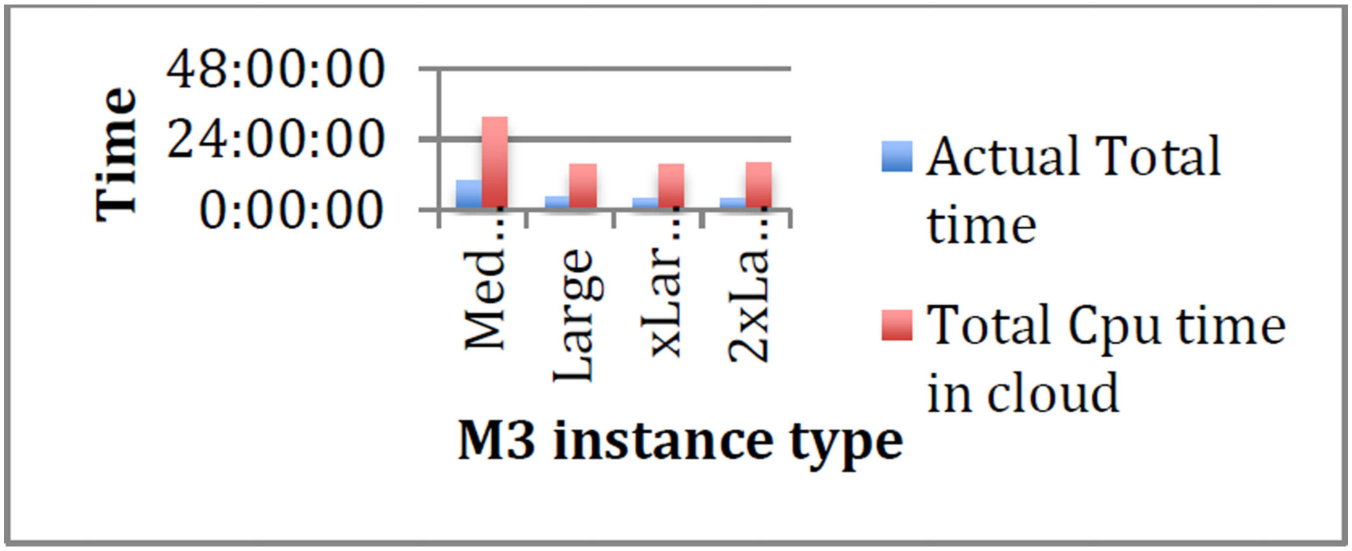
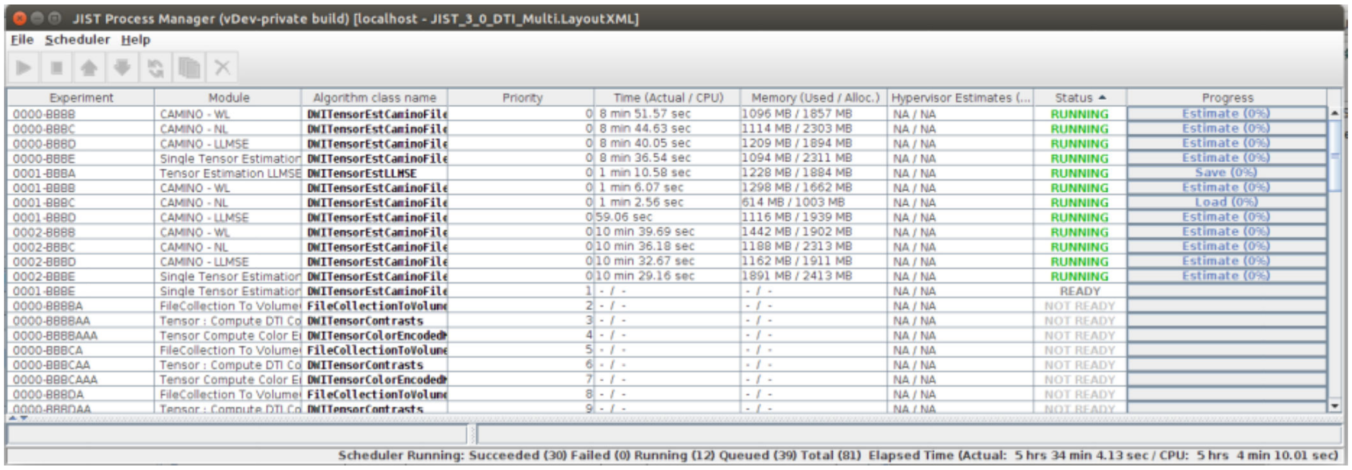


Figure 6.
JIST cloud running time performance with different instance type


JIST Process Manager (vDev-private build) [localhost - JIST_3_0_DTI_Multi.LayoutXML]

Experiment	Module	Algorithm class name	Priority	Time (Actual / CPU)	Memory (Used / Alloc.)	Hypervisor Estimates (...	Status	Progress
0000-BBBB	CAMINO - WL	DWITensorEstCaminoFile	0	8 min 51.57 sec	1096 MB / 1857 MB	NA / NA	RUNNING	Estimate (0%)
0000-BBBC	CAMINO - NL	DWITensorEstCaminoFile	0	8 min 44.63 sec	1114 MB / 2303 MB	NA / NA	RUNNING	Estimate (0%)
0000-BBBD	CAMINO - LLMSE	DWITensorEstCaminoFile	0	8 min 40.05 sec	1209 MB / 1894 MB	NA / NA	RUNNING	Estimate (0%)
0000-BBBE	Single Tensor Estimation	DWITensorEstCaminoFile	0	8 min 36.54 sec	1094 MB / 2311 MB	NA / NA	RUNNING	Estimate (0%)
0001-BBBA	Tensor Estimation LLMSE	DWITensorEstLLMSE	0	1 min 10.58 sec	1228 MB / 1884 MB	NA / NA	RUNNING	Save (0%)
0001-BBBB	CAMINO - WL	DWITensorEstCaminoFile	0	1 min 6.07 sec	1298 MB / 1662 MB	NA / NA	RUNNING	Estimate (0%)
0001-BBBC	CAMINO - NL	DWITensorEstCaminoFile	0	1 min 2.56 sec	614 MB / 1003 MB	NA / NA	RUNNING	Load (0%)
0001-BBBD	CAMINO - LLMSE	DWITensorEstCaminoFile	0	59.06 sec	1116 MB / 1939 MB	NA / NA	RUNNING	Estimate (0%)
0002-BBBB	CAMINO - WL	DWITensorEstCaminoFile	0	10 min 39.69 sec	1442 MB / 1902 MB	NA / NA	RUNNING	Estimate (0%)
0002-BBBC	CAMINO - NL	DWITensorEstCaminoFile	0	10 min 36.18 sec	1188 MB / 2313 MB	NA / NA	RUNNING	Estimate (0%)
0002-BBBD	CAMINO - LLMSE	DWITensorEstCaminoFile	0	10 min 32.67 sec	1162 MB / 1911 MB	NA / NA	RUNNING	Estimate (0%)
0002-BBBE	Single Tensor Estimation	DWITensorEstCaminoFile	0	10 min 29.16 sec	1891 MB / 2413 MB	NA / NA	RUNNING	Estimate (0%)
0001-BBBE	Single Tensor Estimation	DWITensorEstCaminoFile	1	- / -	- / -	NA / NA	READY	
0000-BBBBA	FileCollection To Volume	FileCollectionToVolume	2	- / -	- / -	NA / NA	NOT READY	
0000-BBBBAA	Tensor : Compute DTI Co	DWITensorContrasts	3	- / -	- / -	NA / NA	NOT READY	
0000-BBBBAAA	Tensor Compute Color E	DWITensorColorEncodedM	4	- / -	- / -	NA / NA	NOT READY	
0000-BBBCA	FileCollection To Volume	FileCollectionToVolume	5	- / -	- / -	NA / NA	NOT READY	
0000-BBBCAA	Tensor : Compute DTI Co	DWITensorContrasts	6	- / -	- / -	NA / NA	NOT READY	
0000-BBBCAAA	Tensor Compute Color E	DWITensorColorEncodedM	7	- / -	- / -	NA / NA	NOT READY	
0000-BBBDA	FileCollection To Volume	FileCollectionToVolume	8	- / -	- / -	NA / NA	NOT READY	
0000-BBBDAA	Tensor : Compute DTI Co	DWITensorContrasts	9	- / -	- / -	NA / NA	NOT READY	

Scheduler Running: Succeeded (30) Failed (0) Running (12) Queued (39) Total (81) Elapsed Time (Actual: 5 hrs 34 min 4.13 sec / CPU: 5 hrs 4 min 10.01 sec)

Figure 7.

Maximum 12 jobs are running at once

Table-2

Multi-slice DTI experiment layout top-down structure (one subject).

Job Id	Description	Dependent node	Time estimation
T1	Extract b values and gradient table		Fast
T2	Make a scheme file	T1	Fast
T3	Registration	T1	Slow, 1.25 hrs/subject
T4	Tensor fitting-CAMINO-WL	T3	Slow, 15 mins/subject
T5	CAMINO-WL DTI contrast	T4	Fast
T6	CAMINO-WL DTI Color encode map	T4	Fast
T7	Tensor fitting- CAMINO-NL	T3	Slow, 2.5 hrs/subject
T8	CAMINO-NL DTI contrast	T7	Fast
T9	CAMINO-NL Color encode map	T7	Fast
T10	CAMINO-LLMSE	T3	Slow, 15 mins/subject
T11	CAMINO-LLMSE DTI contrast	T10	Fast
T12	CAMINO-LLMSE encode map	T10	Fast
T13	Single tensor estimation	T3	Slow, 1.25 hrs/subject
T14	Single tensor estimation DTI contrast	T13	Fast
T15	Single tensor estimation color encode map	T13	Fast

Table-3

Case 1 result summary

Instance Type	**Total machines**	**Weight value *A***	**Estimation time T_A (hr:min:sec)**	**Actual Total time (hr:min:sec)**	**Total Cpu time in cloud (hr:min:sec)**	**Total Expense ($)**
Medium (Cpu:1, Mem: 3.75G)	12	0.4	9:22:00	9:57:12	31:27:31.24	8.04
Large (CPU:2,Mem:7.5G)	12	0.9	4:10:00	4:42:40	15:38:8.35	7.98
xLarge (CPU:4, Mem:15G)	12	1	3:45:00	4:09:57	15:34:49.89	15.96
2xLarge (CPU:8, Mem:30G)	12	1	3:45:00	4:07:13	15:49:18.16	31.92

Table-4

12 Large instances v.s. 6 Large instances.

Number of Large instance	**Weight value *A***	**Estimation time T_A (hr:min:sec)**	**Actual Total time (hr:min:sec)**	**Total Cpu time in cloud (hr:min:sec)**	**Expense($)**
12	0.9	4:10:00	4:42:40	15:38:8.35	7.98
6	0.9	4:26:00	5:01:04	15:49:35.8	4.82