CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. 2000; **00**:1–7 *Prepared using* cpeauth.cls *[Version: 2002/09/19 v2.02]*

Advancing Next-Generation Sequencing Data Analytics with Scalable Distributed Infrastructure

Joohyun Kim[1],Sharath Maddineni[1], Shantenu Jha[*2,1]

[1] *Center for Computation and Technology, Louisiana State University, LA, USA*
[2] *Rutgers University, Piscataway, NJ 08854, USA*

SUMMARY

With the emergence of popular next-generation sequencing (NGS) based genome-wide protocols such as ChIP-Seq and RNA-Seq, there is a growing need for research and infrastructure to support the requirement of effectively analyzing NGS data. Such research and infrastrucutre does not replace but complements algorithmic advances developments in analyzing NGS data. We present a runtime-environment, Distributed Application Runtime Environment (DARE), that supports the scalable, flexible and extensible composition of capabilities that cover the primary requirements of NGS-based analytics. In this work, we use BFAST as a representative standalone tool used for NGS data analysis and a ChIP-Seq pipeline as a representative pipeline-based approach. analyze the computational requirements. We analyze the performance characteristics of BFAST and understand its dependency on different input parameters. The computational complexity of genome-wide mapping using BFAST, amongst other factors depends upon the size of a reference genome and the data size of short reads. Characterizing the performance suggests that the mapping benefits from both scaling-up (increased fine-grained parallelism) and scaling-out (task-level parallelism – local and distributed). For certain problem instances, scaling-out can be a more efficient approach than scaling-up. Based upon investigations using the pipeline for ChIP-Seq, we also discuss the importance of dynamical execution of tasks.

KEY WORDS: Next-Generation Sequencing Data Analytics, Science Gateway, BFAST, Mapping, ChIP-Seq, Human Genome, Burkerholderia Glumae, Mouse Genome, Runtime Environment, Distributed Computing, Simple API for Grid Applications (SAGA), Pilot-Job Abstraction

1. INTRODUCTION

High-throughput genome sequencing techniques provided by Next Generation Sequencing (NGS) platforms are changing biological sciences and biomedical research dramatically[**? ?**]. These techniques have led to a broadening of sequencing capabilities and access to comprehensive genome-wide information at increasingly lower costs compared to previous

Received
Revised

sequencing techniques (such as those based on Sanger sequencing) [**? ? ? ? ? ?**]. Thanks to advances in evolutionary protocols utilizing the sequencing technologies such as ChIP-Seq (chromatin immunoprecipitation followed by sequencing) and RNA-Seq, high-throughput sequencing techniques have become essential methodologies in studies of cell development and differentiation[**? ? ? ? ?**]. Indeed, the resulting influx of biological information about the genome-wide organization for regulatory protein-DNA interactions and transcription dynamics reveal underlying mechanisms of gene expression and regulation in living cells. This has the potential to lead to remedies for various diseases such as cancer, infectious diseases, and dysfunctional diseases caused genetically or by aging[**? ? ? ?**].

It is expected that data-sets of interest will increase by several orders of magnitude as the number of genomes that need to be sequenced together increases exponentially. For example, genome-wide variation studies require a statistical number of genomes for one single species as shown in the recent 1000 genome project and human genome studies[**? ? ? ? ?**]. Although high-throughput techniques enjoy extremely high coverage of target genome regions, current "deep sequencing" technologies adopted by NGS platforms such as Illumina GA II and Applied Biosystems SOLiD are limited to generating only short sequence reads, generally less than 100 hundred nucleotides[**?**]. Consequently, it has become a computational challenge to map these high volume short reads onto a reference genome or de novo assembly that are needed as the first step for any genome-wide studies[**? ? ? ? ? ?**].

To address such challenges, there have been algorithmic advances and many software tools have become available[**? ? ? ? ?**]. However, compared to the algorithmic advances and new software tools, the development of a well-architected integrated infrastructure – the software & services, data management capabilities, has received less attention.

As we will demonstrate, NGS analysis is not a simple or singular compute or data-intensive application, but an interesting mix of high-end and high-throughput computing with data-intensive requirements; thus a scalable and integrated infrastructure is required. The need for multiple tools, each with varying CI requirements appropriate scalable and flexible architecture for the infrastructure becomes critical. In analyzing the suitability of distributed cyberinfrastructure for NGS, we hope to contribute to the fundamental question: what is the appropriate integrated compute and data infrastructure to address the existing requirements of NGS analytics now and appearing in the future?

We analyze the computational complexity of BFAST [**? ?**] – one of the latest generation of alignment tools. The determinants of its performance were investigated for two exemplary genomes, human genome and a microbial genome *Burkerholderia Glumae*[**?**]. Secondly, having understood the computational requirements, and the importance of both scaling-up and scaling-out with the alignment tool, we present a pipeline for ChIP-Seq data analysis. Based on our investigation with these two examples, we came up with the initial design objectives and a prototype of DARE-NGS to support a range of genome-wide analytics.

The Distributed Application Runtime Environment (DARE) framework [**?**], is a runtime infrastructure that has been shown to enable the seamless utilization of heterogeneous distributed computing resources [**? ? ?**], including high-performance computing (HPC) grids and cloud environments.

DARE provides an effective framework to build a scalable infrastructure supporting a wide range of applications, abstractions and execution requirements, including those for the

analytics required for NGS data. DARE-NGS builds upon existing capabilities of DARE, and is designed on the organizing principle that the bulk of NGS-analytics can be categorized into three types of services – Type I, II and III (see Table **??**). Type I represents a service adequate for stand-alone tools, but transforms those tools as dynamic applications with a scalable and flexible runtime environment. Types II and III services provide integrated pipelines and workflow composition of tools respectively. The main design strategy of the DARE framework is to separate the application logic from a target tool(s), consequently resulting in easily extensible, interoperable and flexible deployment of new tools.

This paper is organized as follows. In §2, we introduce BFAST and a ChIP-Seq pipeline, and characterise their computational and data requirements. In §3, BFAST performance is characterized and analyzed using local and multiple distributed resources (HPC grids and clouds). The execution of the ChIP-Seq pipeline is presented as an example of a Type II service and the importance of flexible and dynamic pipelines is established. In addition to a summary, we conclude with a discussion of future directions which we will pursue.

2. Characterizing BFAST and ChIP-Seq Pipeline

Before investigating NGS data analysis and their infrastructural requirements for mapping and ChIP-Seq, we discuss a runtime environment that allows distributed and dynamic execution of standalone tools, pipelines, and workflows, collectively referred to as applications. We categorize the resulting applications into three types of services as summarized in Table **??**.

Three Types of Services for NGS Data Analytics: We will examine two specific software tools, BFAST for the mapping application and a new pipeline for ChIP-Seq data analysis. For the ChIP-Seq pipeline, we used MACS as the peak caller, along with three different mappers, BFAST, BWA, and Bowtie. Note that these two examples show how existing computational tools can be used as Type I/II services. In this paper, we will focus on Type I and II, as the same underlying concepts can be generalized to Type III services.

With Type I services, we focus on how the enhancement of existing individual software tools is achieved with scalable infrastructure and supporting dynamic execution patterns, whereas with Type II services, we will emphasize an example of the development of a pipeline service by integrating multiple tools. For evolutionary techniques such as RNA-Seq and genome wide mapping of DNA-protein interactions (ChIP-Seq) or methylation patterns, pipeline approaches have been playing increasingly important roles, implying that the need for Type II services is growing fast.

2.1. BFAST: A Mapping Tool for Type I Service

Before we propose a scalable, extensible infrastructure to support data-analytics on a range of problem-instances, it is critical to characterize the computational and data dependencies of the application and corresponding software tools. The overall execution of BFAST, or any other mappers that align short reads onto a reference genome, is affected by the specific biological problem at hand – such as types of target reference genomes and high-throughput sequencing

Service Type	Type I	Type II	Type III
Service Description	Standalone Single Tool	Dynamic Pipeline or Multiple Tools	Dynamic Workflow-based
Example Application	Mapping	ChIP-Seq, RNA-Seq	ChIP-Seq, RNA-Seq
Example of Existing Tools	BFAST, BWA, Bowtie, ABySS	MACS, TopHat, Cufflinks, GATK TRANS-ABySS, Scripture, Hydra	N/A

Table I. Three kinds of services for dynamic applications developed for NGS data analytics.

techniques and protocols, as well as computational attributes such as data structure and formats and the ability to exploit the underlying architecture. Here, we present various runtime options for BFAST that ultimately permit performance tuning; we also present measurements of the compute and data requirements for the problem instance of interest.

2.1.1. BFAST Static and Runtime Options

BFAST requires a reference genome sequence and NGS platform generated short reads initially, and produces mapping results of tens or hundreds of millions of short reads onto a target reference genome. Notable key common features of BFAST that are generally shared among diverse genome-wide analysis applications include: i) it requires input data containing sequence information of a reference genome or short reads from NGS platforms, ii) production of output information that is generally written with a format that is in turn injected to another tool as input.

There can be large variation in the data volume – input, output, and/or temporary files, often requiring that BFAST should support different (local versus distributed) execution modes. BFAST supports classic space-time tradeoff capabilities, i.e., memory and disk usage versus time-to-completion. BFAST can be run using multi-threads (ideally on a multi-core machine).

Specifically, BFAST supports multi-threading and a "low memory" option for the 'bfast match' step, which finds Candidate Alignment Locations (CALs) of each short read in indexes of a reference genome. The memory option aims provides the ability to split the index files into several files, and thereby facilitating low (peak) memory consumption. The number of threads is determined by the parameter (`-n`); the `-d` option is used to split the index into multiple parts in order to support low-memory computation

Note that the `-d` option is useful for specific architectures (with low memory), but as we present later, it requires more computing time. The number of index files generated with the `-d` flag is N_i, which equals $n_m \times 4^d$. Here, 4^d index files are generated by splitting one original index file and n_m is the number of masks for indexing, and usually fixed as 10 for our work. For example, $d = 0$ creates 10 index files, whereas $d = 1$ creates 40 index files, i.e. four index files are generated for each mask and processed sequentially.

Taken together, BFAST can support varying degrees and types of parallelism or multi-threading, and thus is suitable for modern (processor) architectures. Nonetheless and more

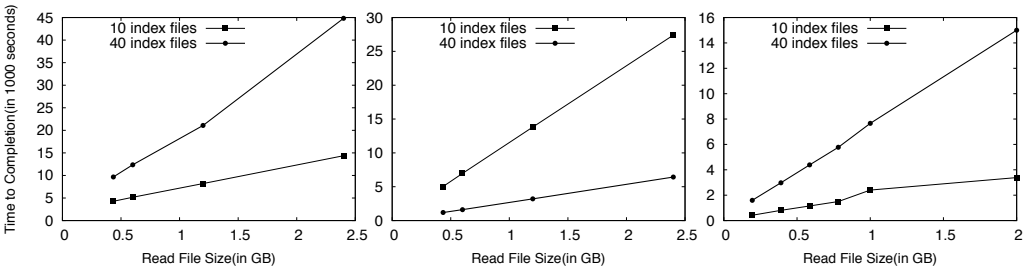

Figure 1. The time-to-completion of the 'bfast match' step is measured as a function of the size of short read files. The different lines represent two different memory options, which results in different number of index files (40 vs 10). Results for three different reference sequences for exome analysis are provided here: (i) the entire human genome (hg18) (left), chromosome 21 only (middle) and, B. Glumae (right). The number of threads is set to 12.

Genome (Release ID)	Human (hg18)	B. Glumae (BGR1)	Mouse (mm9)
Reference Genome			
# of Base Pairs (bp)	3 Gbp	7.3 Mbp	2.6 Gbp
Genome Structure	23 (pairs of chr)	2 (chr) & 4 (pl)	21 (pairs of chr)
NGS Data			
Type of Analysis	Exome Analysis	Resequencing	ChIP-Seq
Sequencing Platform	ABI SOLiD	Illumina GA2	ABI SOLiD
Read Length	50 bp	36 bp	36 bp
Sequencing Data (fastq)	8.7 GB	5.4 GB	5.7 GB (treat) & 6.5 GB (control)
BFAST			
Minimum Disk Space	approx. 200 GB	approx. 30 GB	approx. 84 GB (chr 19)
Index Files Volume	approx. 128 GB	approx. 0.447 GB	approx. 3.1 GB (chr 19)

Table II. Summary of reference genome sequences, NGS data, and required disk space for running BFAST. The mouse genome used for ChIP-Seq data analysis is converted in nucleotide space in spite of the fact that sequencing data were obtained with the ABI platform. For ChIP-Seq, two short reads data sets (control data and treat) are needed. All read data are single-end.

importantly, it is valid to ask if there remains other bottlenecks (eg data flow/access) in the utilization of heterogeneous distributed infrastructure? If the answer is found to be positive, it is natural to ask whether appropriate runtime environments can be developed in order to overcome these limitations?

2.1.2. Computational Requirements for BFAST

In Table **??**, we summarize the typical values of the parameters upon which the time-to-completion (T_C) depends. We consider two genome data sets, i.e. human (eucaryote), and a microbe (procaryote), Burkerholderia Glumae[**?**]. The two genomes differ in the size and the genome structure of reference genomes, and types of sequencing protocols. Importantly, the two different genomes display widely different data sizes in terms of the reference genome sequence, short reads data, and required disk-space for carrying out the mapping with BFAST.

To investigate the computational requirements, we first measure the time-to-completion for 'bfast match' step while varying the size of a read file. The results are shown in Fig. **??**. Here, we present the T_C of exome analysis with two different reference genomes, the entire human genome (hg18) and one of its chromosomes, chromosome 21 (hg18-chr21), along with the whole genome resequencing with a microbe, B. Glumae[**?**] (see Table **??**). For each case, we compare two different options for the `-d` flag.

As shown in Fig. **??**, time-to-completion scales linearly as the size of read files increases. This suggests that if the read files can be broken up into smaller fragments and which can be processed concurrently, there is the possibility of a speed-up. This points to the potential of task-level concurrency.

Secondly, the results with the memory option ($d = 1$) generates 40 index files (factor of four increase in the number of files compared to the option with $d = 0$) indicates that there is approximately a factor of four difference in the time-to-completion. We calculated the ratios of slopes between 40 and 10 index files and those are 3.50 (hg18), 4.25 (hg18-chr21), and 4.46 (B. Glumae), respectively. Although there are many factors including the complex structure of the index files that effect the slow-down with the $d = 1$ option, the main reason is that such low memory option requires four CAL finding processes for each read that are conducted successively with four different index files as indicated in the publication[**?**]. In other words the peak memory consumed is dependent on the size of the index files (which decreases as the number of index files increases), whilst the runtime is dominated by the size of the read-file (linear dependence).

Finally, we observe that the entire human genome required only a two fold increase in T_C compared to the mapping calculation with only chromosome 21. We note that disk space and memory requirements can in some cases prohibit the use of a large reference genome. For example, the human genome requires about 200 GB disk space and 16 GB memory and in fact, not all systems can (see Table **??**) be used with the whole human genome as a reference genome. In such cases, the task-level concurrency is a possible solution. Overall, the three cases shown in Fig. **??** highlight the diverse biological contexts along with the computational requirements of BFAST, leading to different optimal strategies for distributed parallel executions.

2.1.3. Characterizing Data Requirements for BFAST

Generally speaking, mapping should deal with data for a reference genome, short reads, and processed data generated in each step. Particularly, the temporary data generated during 'bfast match' step needs careful attention due to their significant size and more importantly

Case	Read Files	Index Files	Size of Temp File	# of Temp Files	Approx. Disk-space
I	40	40	105MB	16	67 GB
II	20	40	220MB	16	70 GB
III	40	10	105MB	13	52 Gb

Table III. Table showing the required disk-space for operating data, for varying number of read files and index files, when all the read files are processed concurrently. The number of concurrent tasks are equal to the number of read files. The reference genome used is Human Genome (hg18) chromosome 21 with total index file size 1.9 GB and the total size of the read files was 8.9 GB.

Computing Environment	Organization	System Used	Storage Type	Disk-space Limit
HPC Grid	LONI	QB	Disk	58 TB
		Painter/Eric	Disk	100 GB
Cloud	FutureGrid	INDIA	Walrus	230GB
		SIERRA	Walrus	58GB
Workstation	Local	Cyder	Disk	5 TB

Table IV. Specification of two distributed environments. LONI represents Louisiana Optical Network Initiative[**?**]. FutureGrid Cloud[**?**] employs Eucalyptus

a characteristic aspect such that the size depends on how the step is executed, for example, concurrently vs. serially, or centralized storage vs. distributed storage, or the low memory option (i.e. the number of index files).

We estimated the peak disk-space requirement. As shown in Table **??**, the number of temporary files as well as the size of each temporary file change depending upon the configuration of 'bfast match' step such as the number and the size of each read file and the memory option invoked. The memory option changes the number and the size of each index file, and thus the corresponding temporary files are generated differently. For example, cases I and II that are with $d = 1$ and 40 index files require more space for temporary files than case III with 10 index files ($d = 0$) does. Case II needs a bit more space than case I, for it is conducted with a half number of read files but twice size of each read file. In summary, run-time disk-space analysis suggests that data volume might be a major issue for a large genome system.

2.2. ChIP-Seq Pipeline: An example of Type II Service

The ChIP-Seq experiment provides genome-wide mapping of DNA-protein interactions, and obtained information is directly associated with transcription regulation[**? ? ?**]. An understanding of transcription regulation is one of the fundamental goals in biological sciences, which is relevant to virtually all aspects of biological processes in a living cell responsible for cell

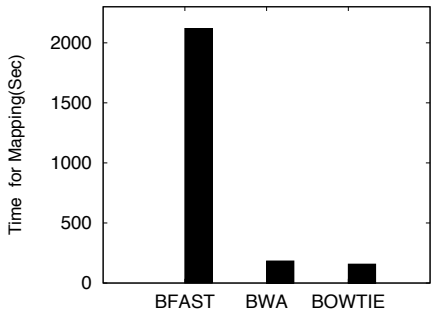

Figure 2. Comparison of time-to-completion for the mapping step with three different mappers – BFAST, Bowtie, and BWA. A short read file size of 100 MB is used.

development, differentiation, host-pathogen interactions, epigenetics, tumorigenesis, and so on. New drug or bio-marker discovery for many devastating diseases including neurodegenerative disorders are examples of anticipated outcomes from such studies[**?**].

ChIP-Seq protocol involves Chromatin Immunoprecipitation (ChIP) followed by sequencing (ChIP-Seq). Following this protocol, the sequenced reads indicate the location of a binding site or a peak where regulatory proteins bind as manifested by the enrichment of reads in such a region. Replacing ChIP-ChIP methods based on micro-array approaches, the ChIP-Seq experiment has rapidly become a major protocol with its genome-wide scale information and higher resolution[**? ?**].

In the first step, the mapping of two data sets, corresponding to the control (if it exists) and the treatment data (the main data with the experimental condition from the ChIP experiment), are carried out separately, and in the second step, a peak caller, which carries out statistical analysis to infer true peaks representing DNA-protein interaction regions. The control data facilitate the construction of a peak model and background calibration, but is optional.

In this work, we use the widely used peak caller – MACS[**?**], for the second step, while three different mappers – BFAST, Bowtie, and BWA, are employed in the first mapping step. These mappers differ in many ways, particularly by their different computational requirements. The two additional mappers, BWA and Bowtie, are widely used as they implement the memory efficient Burrows-Wheeler Transformation, whereas BFAST was found to provide increased sensitivity[**? ?**]. Note that the second step with MACS takes a relatively shorter time than the mapping step and thus we focus on the computational characteristics with the first step.

In Fig. **??**, we present the comparison of the three mappers, BWA, Bowtie and BFAST for the mapping step. Note that the results were obtained by using the pipeline we developed as a Type II service, in which the mapping step was carried out using 256 cores of a HPC cluster (QB) for 128 tasks concurrently.

3. NGS Data Analytics using Distributed, Scalable Architectures

A widely-used tool-box that provides a variety of tools for genome-wide analysis through as a single framework and software package has been recently developed[**?**]. While interesting progress has been made in algorithms, some important requirement are notably missing in existing infrastructural solutions. A significant limitation is that most solutions are tailored for a specific problem instance. But can an infrastructure be designed such that it meets the requirements of a large range of problem instances, both in terms of the size of the data involved and the amount of computing required?

There have been significant recent efforts in utilizing emerging computing environments such as clouds for genome-wide analysis and infrastructure development efforts[**? ? ? ? ? ? ? ?**]. However clouds impose a strong model of data localization. We believe that in order to meet these requirements, the ability to utilize multiple and heterogeneous resources will be inevitable.

3.1. Distributed Application Runtime Environment

We have established that efficient analytics on NGS data is not just a matter of data-management, but also of providing large amounts of computing for analysis. Any solution that facilitates analytics across different problem instances will require a general purpose framework that is able to exploit different types of architecture and platforms.

This could involve supporting fine-grained parallelism of the type that is supported by invoking the multi-threading option of a software tool such as BFAST, or it could involve invoking multiple instances of the tool executing independently but on different read-files associated with the same problem instance. Which approach will be more effective depends upon the specific configuration of the problem instance, i.e., a problem instance may be I/O bound or compute/memory bound.

To achieve the goal of supporting both types of parallelism on heterogeneous distributed computing resources, we have developed the Distributed Application Runtime Environment (DARE) framework[**? ? ?**].

At the core of the DARE framework, is a flexible general purpose pilot-job implementation – BigJob[**? ? ? ? ?**]. In subsequent sections, we will demonstrate via performance numbers the effectiveness of how the DARE framework supports a range of infrastructure, problem instance configurations and size. Such task-level concurrency supports different ways of splitting the data, whilst using the same runtime environment using BigJob; all the parallel tasks of BFAST steps are defined as sub-jobs in BigJobs. It is possible to optimize at multiple-levels, in that different thread/cores configurations are possible for each of the sub-jobs.

DARE has distinctive characteristics compared to other run-time environments [**? ? ? ? ? ? ? ?**], many of which have been developed in the Cloud environment. First of all, most other approaches lack interoperability between different distributed computing environments. This is partly because the application layer is tightly coupled to the runtime environment layer. Secondly, other approaches support a single application or a few applications with limited extensibility. Some tools such as CloudBurst[**?**], CloudBlast[**?**], Crossbow[**?**], and GATK[**?**] have advantages that are specifically confined to Cloud environments; they permit ease of

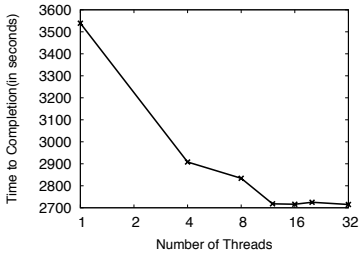

Figure 3. Measuring the time-to-completion as a function of the number of threads. A single node on the machine (Cyder) used for tests has 12 cores and the reference genome is Human genome (hg18) chromosome 21. The low memory option with 40 index files is used for this measurement. The T_C decreases logarithmically, until the number of threads is equal to the number of cores, after which the performance levels off. Similar behavior is observed for different machines with varying number of cores.

utilization of effective programming models such as MapReduce or other software frameworks, whereas others[**?**] focus on the HPC grid environment. Indeed, as we will show in the context of ChIP-Seq, the DARE framework proposes a means of supporting dynamic applications that overcomes the limitations of previous approaches by supporting interoperability between distributed computing environments and by supporting a wide range of applications.

3.2. Understanding BFAST on a Local (Single) Resource

3.2.1. T_C on N_t and N_c

To determine how the performance of the bfast match step varies with the different number of threads (N_t) and cores (N_c) we measure the time-to-completion (T_C) of the bfast match step on a single machine with multiple cores. As shown in Fig. **??**, (T_C) indicates a speed-up until the number of threads becomes equal to the maximum number of cores. Interestingly, the overall performance gain is about 30 %, and the scaling tendency does not reveal any strong pattern but a weak log-linear behavior.

3.2.2. Understanding I/O

We conducted experiments with four different configurations, wherein we varied the number of threads per task and the number of tasks running concurrently such that the total number of cores utilized at any stage is fixed to the number available on the single node – 4 (see Table **??**) The total amount of short-data processed per task varied such that the total amount of short-data processed was a constant.

Case	Read File Size	Threads per Task	# of Tasks	T_C
s1	2.4 GB	4	1	9358 s
s2	1.2 GB	2	2	5290 s
s3	0.6 GB	1	4	9152 s
s4	0.6 GB	4	4	5804 s

Table V. Computing time for the 'bfast match' step on a single node (Eric). Note that all cases have the same amount of total read data, and use ten index files. Under the restriction that the hardware node has 4 cores, the time-to-completion is compared with varying conditions regarding the size of read files, the number of threads for each 'bfast match', and number of the 'bfast match' executed concurrently.

Case s1 provided the starting configuration where 1 (BFAST) task was run with 4 threads; in subsequent configurations (s2 and s3) the number of tasks were increased (up to the number of cores available), with a concomitant decrease in the threads per tasks (so as to be constant at 8). We find that although in general, T_C decreases as the number of tasks increase (s1 to s2) and thus the amount of data-processed per task decreases, that is not universal. Specifically in going from s2 to s3, there is an increase in T_C. The exact cause for such behavior is difficult to discern in the absence of performane profiling, but circumstantial evidence points to read-contention on index files along with an I/O bottleneck.

In the case of s4, 4 BFAST tasks were also run on a single node; the performance was better than for s3. This can be expected because although the four BFAST tasks were trying to access the same set of index files for processing, the multi-threading option was used. In this case all LONI machines have a Lustre filesystem for accessing the input data. Because of increase in access of same files there has been a unpredictable behavior in time-to-completion (s3). This was consistent with observation on other local and cloud resources.

3.3. Characterizing BFAST (mapping) on Distributed Resources

In order to demonstrate the flexibility of DARE and its ability to support both fine-grained parallelism and task-level parallelism over multiple and distinct environments, we utilize a HPC-grid (LONI) and a Cloud resource (FutureGrid) (see Table **??** for details of the infrastructure).

3.3.1. Using BFAST on HPC Grids

The first two configurations – case g1 and g2 in Table.**??** demonstrate how DARE can support large-scale task-concurrency. Specifically, in g1 & g2 all parameters and conditions are kept similar, but the number of BFAST tasks (sub-jobs) are varied and thus the number of cores utilized. The individual processing time of each BFAST task decreases in proportion to the data-file size; interestingly, as can be seen, with twice as many sub-jobs the time taken decreases by half. Therefore, the increase in the coordination cost of multiple BFAST tasks is negligible,

Case	Read File Size	Threads per Task	# of Tasks	BigJob Size Cores(Nodes)	T_C
g1	0.209 GB	2	40	80(10)	3966 s
g2	0.435 GB	2	20	40(5)	8031 s
g3	0.209 GB	2	40	12(3)	25807 s
g4	0.435 GB	2	20	12(3)	23872 s
g5	0.209 GB	2	40	80(20)	1111 s
g6	0.435 GB	2	20	40(10)	2096 s

Table VI. Performance comparison for different parallel configurations using SAGA-BigJob on a HPC-Grid (LONI). One BigJob is submitted with the number of sub-jobs, where each sub-job is a BFAST task. The total (read) data size is the read-file size multiplied by the number of tasks (which is equal to the number of read-files); which is a constant for g1-g6. Cases g1, g2 and g3,g4 and g5,g6 are conducted on QB, Painter and Eric respectively. The cases g1, g2, g3, g4 are use 40 index files of a Human Chromosome 21. Note that g5 and g6 are the results with 10 index files; g6 is specifically carried out to provide a direct comparison to c5 (on a cloud resource as in the following Table **??**)

which indicates a simple, but important test of scalability (with the number of tasks) of the DARE framework.

In cases g3 and g4, the number of BFAST tasks (sub-jobs) is greater than the number of cores available at any one instance; thus multiple generations of tasks have to be staged in as cores become available (when earlier tasks finish). For g3, six generations of tasks have to be completed; comparing g3 with g1 there is an approximate factor of six difference in T_C. Similarly T_C for g4 is a factor of six greater than g2 as would be expected if the coordination overhead of staging in tasks was not too great. This suggests that the DARE framework not only supports efficient concurrent task execution, it also supports efficient execution over a range of BigJob/resource sizes.

For g5 and g6 the number of index files is 10; to be contrasted with 40 for g1-g4. However, in spite of a different index file number, the linear scaling behavior exists. As discussed in §2.2.1, the time-to-completion has a linear dependence on the index files (memory-time trade-off); once this is taken into account the total time-to-completeness is consistent with g1-g4.

3.3.2. Using BFAST on Clouds

In the previous sub-section, we established the ability of DARE to support concurrent task execution by establishing the scale-out to a large number of cores/resources. Although we did not use physically distributed resources to validate their performance, we used logically distributed tasks and have in earlier work established performance over distributed HPC grids[**?**]. The question then arises: can DARE support similar capabilities of scaling on clouds without sacrificing the ability to utilize fine-grained parallelism as needed? We attempt to address this question in the remainder of this section.

As mentioned, the cloud infrastructure used was the Eucalyptus based IaaS resources on the NSF-sponsored FutureGrid[**?**]. Resource descriptions are summarized in Table **??**.

Case	Read File Size	Threads per Task	# of Tasks	Total Cores	# of VM's	T_C
c1	0.435 GB	2	4	8	4	1492 s
c2	0.435 GB	2	4	8	1	1149 s
c3	1.74 GB	8	1	8	1	4528 s
c4	0.435 GB	2	20	40	1	1137 s

Table VII. The number of index files used for this measurement is 10. In all cases the input and output data resides on VM itself. VM in case c1 is of type m1.large whereas c2, c3, c4 are of type c1.xlarge

Case c1 and c2 have the same workload, however c2 uses an instance type c1.xlarge, whilst c1 uses a m1.large instance type. In the case of c1 four different BigJob agents on four VM's were launched, as opposed to one BigJob agent on a single VM in case of c2.

For exactly the same workload, the T_C for cases c1 and c2 are shown in Table **??**. We find that even though the number of cores and number of BFAST tasks employed are the same (8 and 4 respectively), the T_C for c2 is lower than that for c1; this is because there exists an overhead of marshaling 4 BFAST tasks into 4 VMs compared to 1 VM. Additionally, c1.xlarge is a more powerful instance than m1.large.

In contrast, c3 represents the configuration when a single large instance (c1.xlarge) is used for a single BFAST instance but using multiple threads; interestingly the T_C is larger than c1 or c2. We attribute this to the fact that increasing the number of threads increases the performance only logarithmically, whilst increasing the number of concurrent tasks executed (c1 and c2) has a linear performance improvement. The increase in T_C for c3 now arises due to the fact that the size of the read file that is an important determinant of performance is now 4 times as larger than c1 or c2.

3.4. The need of flexibility, extensibility, and scalability in a ChIP-Seq pipeline

In Fig. **??**, we compare the number of predicted peaks using the developed pipeline with three different mappers. These calculations were carried out without considering the control data and the reference genome sequence is chromosome 19 of mouse genome (mm9). MACS was used for the peak calling step for these calculations. The first step, the mapping process using a mapper was implemented to support the distributed mode for which 128 files of ChIP-Seq reads were generated from the full data set and processed concurrently. The second step, the peak calling, which takes significantly less time (and thus support for concurrent execution is not critical) was set to execute after gathering all mapping results into the DARE server. Using distributed resources, the three different peak calling calculations can be conducted at the same time and various execution scenarios are easily implemented with DARE.

As shown Fig. **??** the number of predicted peaks, or the location of binding sites of biological interests, peak calling using the same ChIP-Seq data was dependent on the mapper. We observed that not only the varying number of predicted peaks but also significant difference in statistical measures for a predicted peak such as False Discovery Rate were observed. The

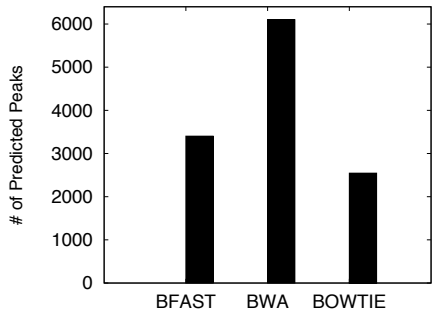

Figure 4. Peak calling results for three different ChIP-Seq pipeline scenarios. A different mapper is used for the first step. For these calculations, the same treatment read data are mapped onto the chromosome 19 of mouse reference genome (mm9) and MACS is used as the second step.

resulting difference is in fact not surprising since the mapping statistics already indicate a dependency on the specific chosen mapper. For example, we found that with the treatment data, 8,375,179 reads were mapped by BFAST whereas 3,104,010 reads and only 277,427 reads were mapped by BWA and Bowtie, respectively. The general tendency that BFAST maps more reads than the other two mappers was observed by others and us with different data sets[**? ?**].

Recent review articles comparing peak callers reported difference in results were due to different algorithms and statistical analysis strategies employed[**? ?**]. Considering the complex nature of peak calling process, the practical validation of a chosen software tool for peak calling is non-trivial and often out of the question especially for ordinary researchers. Here, according to our results, we find that the consequence of the mapping step is another factor affecting sensitivity of peak calling.

In summary, we presented a case for requiring a flexible pipeline in which multiple alternative tools are easily employed and scalable executions for increased computing requirements are seamlessly supported. This is because biologists are likely to seek the assessment of their findings with final experimental validation, implying that they might favor a practical means to have all candidate peaks from multiple combination of tools in the two step ChIP-seq pipeline. Our ChIP-Seq pipeline is an example of DARE Type II service, and provides a practical solution for the flexible deployment of a range of software tools on given distributed reources.

4. Conclusion and Future Work

The challenges of Next Generation Sequencing are significant and pervasive. BFAST, has emerged as one of many important analytical tools for NGS, is a sophisticated application that

has the capability to utilize advanced architectural features – in particular at the processor level, such as multi-threading and support for low memory run-time configurations. However, by analyzing the human genome, we find that more often than not, in order to perform analysis at the scales required, additional support for concurrent execution is required, whilst exploiting the fine-grained parallelism. We also observed, that for BFAST – which is a data-intensive application, the performance bottleneck often becomes I/O; we establish that in order to overcome the I/O induced bottleneck, any effective run-time environment must support both local and distributed execution. In other words, a sophisticated run-time environment that addresses the challenges for both local and distributed environments is required.

This paper represents the initial steps in the design and development of a general-purpose, scalable and extensible infrastructure to support next-generation (gene) sequencing data-analytics. To the best of our knowledge the approach in this paper is unique in that we analyze both the data-intensive and the computational requirements of typical analytics performed on the data, which in turn motivates the architecture and implementation of the infrastructure.

We have analyzed the requirements which are valid for a broad spectrum of problem instance sizes; however, we have demonstrated the capabilities and effectiveness of our architecture for a reduced problem instance. The immediate next step is to demonstrate the effectiveness for larger problem instances, e.g., genome wide variation studies with many individual human-genomes, through both the scale-up and scale-out of the infrastructure. Additionally, we will extend our framework to develop and support multi-stage workflow.

We have begun implementing the next steps and features for this project. In an attempt to provide these advanced capabilities to the community, we are working on NGS Data Analytics Science Gateway that will abstract many of the infrastructure details and provide "autonomic" capabilities, i.e., map the specific problem instance to the appropriate backend infrastructure. Mapping with BFAST and the pipeline discussed in this work are freely available via the DARE-based gateway (DARE-NGS available at `http://dare.cct.lsu.edu`). Type II services and Type III workflows will be added soon.

Acknowledgements

The project described was partially supported by Grant Number P20RR016456 from the NIH National Center For Research Resources. We also acknowledge Ole Weidner and Le Yan for useful performance related discussions, Diana Holmes for sharing her experience with mapping using BFAST, Jong-Hyun Ham for allowing us to use B. Glumae genome sequences, and Chris Gissendanner for discussions on ChIP-Seq analysis. Computing resources were made possible via NSF TRAC award TG-MCB090174 and LONI resources. This document was developed with support from the National Science Foundation (NSF) under Grant No. 0910812 to Indiana University for "FutureGrid: An Experimental, High-Performance Grid Test-bed."