Porting the MG-RAST metagenomic data analysis pipeline to the cloud

Andreas Wilke, Jared Wilkening, Elizabeth M. Glass, Narayan L. Desai, and Folker Meyer

Argonne National Laboratory

Abstract

Computational biology applications typically favor a local, cluster-based, integrated computational platform. We present a lessons learned report for scaling up a metagenomics application that had outgrown the available local cluster hardware. In our example, removing a number of assumptions linked to tight integration allowed us to expand beyond one administrative domain, increase the number and type of machines available for the application, and improve the scaling properties of the application. The assumptions made in designing the computational client make it well suited for deployment as a virtual machine inside a cloud. This paper discusses the decision process and describes the suitability of deploying various bioinformatics computations to distributed heterogeneous machines.

1. Introduction

Next-generation DNA sequencing instruments have made large-scale sequencing widely available to users. While sequencing was dominated by large-scale sequencing centers as recently as a few years ago, the advent of novel sequencing machines (1,2) have begun a process referred to as the democratization of sequencing. As this process continues, the costs and availability of sequencing machines will improve broadly. Meanwhile, the data production rates of these sequencing instruments are growing rapidly, leading to a crisis in data analysis.

Next-generation sequencing instruments are used in a variety of areas; however, the area that has seen the most dramatic change in computational requirements is the analysis of shotgun metagenome data, or metagenomics (1-3). Also known as community genomics, metagenomics is the study of genomic data from environments of unknown biological composition. Metagenomics analysis techniques are routinely used to study microbial communities in a variety of environments. For example, the TARA expeditions (http://oceans.taraexpeditions.org), the Terragenome project, Global Ocean Sampling (4), and the Human Microbiome Project (5) all make heavy use of metagenomic analysis techniques.

Over the past two decades the amount of public DNA sequence data, generally from conventional genomic projects, has grown dramatically, doubling about every 14 months (see Figure 1). Early experience with metagenomics projects suggests that metagenomics sequence data will have a substantially shorter doubling time (1). While traditional genome sequencing requires establishing an in vitro culture of the organism to produce sufficient DNA material for sequencing, metagenomics relies on direct sequencing of

environmental samples without the intermediate in vitro step. Removing the culturing step has led to a flurry of metagenome projects and has brought sequence-based biology into a new phase as the major challenge shifts from generating to *analyzing* sequences.

In this paper, we describe the MG-RAST service, including its architecture and computational bottlenecks. We discuss the process through which we adapted MG-RAST to make use of cloud computational resources. We also describe the technical challenges we faced, as well as unexpected issues we encountered.

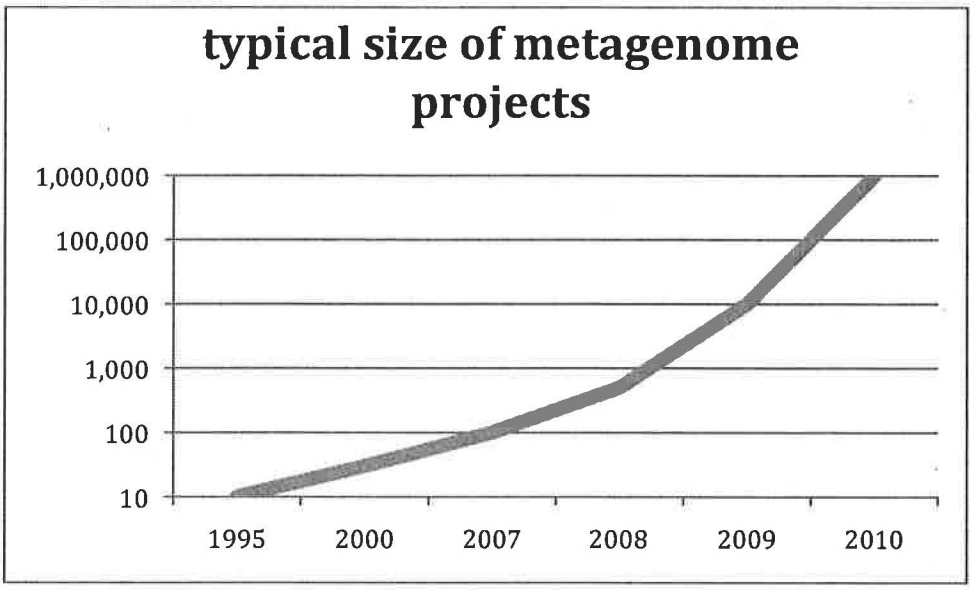

Figure 1: Typical size of metagenome data sets (in megabases) over 15 years. In the 1990s metagenomic data sets consisted of a few megabases; this number grew to a few hundred megabases in the early 2000s. With the advent of 454 and Illumina sequencing, data sets now routinely reach hundreds of gigabases (10^6 bases), and it is likely that terabase (10^9 bases) will begin to appear in 2011.

1.1 MG-RAST

The Metagenomics RAST server, or MG-RAST (7), allows public upload and analysis of data via its web portal. MG-RAST has been operating since 2007 and now has over 2,000 users, from 30 countries. As the sizes of metagenomic data sets have grown, so have the computational costs of the analysis. Our goal over the past several years has been to keep pace with the increases in sequencing data set sizes, so that users can continue to analyze similar numbers of data sets, albeit with higher fidelity provided by newer, larger data sets. To achieve this goal, we have had to scale up the computational capacity of MG-RAST.

In order to provide a frame of reference for both the absolute demand for computational capacity and some relative growth rates, we have performed a substantial amount of benchmarking for MG-RAST (8). During this work, we determined that the analysis of 1 gigabase of data using the MG-RAST v2 protocols consumed approximately 2,000 core

hours on Intel Nehalem machines. A majority of this time is spent performing sequence similarity analysis. The data growth rates are far more troubling; we have observed a growth from 17 gigabase data sets to well over 300 gigabases in the past 18 months. This growth rate is unlikely to slow in the next few years. Taken together, these factors require both a quickly growing computational infrastructure and approaches to effectively harness it.

In mid-2009 a moderately sized cluster of 32 commodity machines supported MG-RAST analysis. With this scale of resources, our pending workload far outstripped our capability to analyze the data. Our analysis techniques had already been optimized; the majority of the cost came from the similarity computation, performed by NCBI's BLASTX (9) code. Unlike large-scale comparison with DNA databases where a variety of other codes exist (e.g., (4-6)), there were no algorithmic alternatives to the computationally expensive BLAST code. Furthermore, even with significant improvements in algorithms, data set size and user base growth both require considerable expansion of the underlying computational platform.

While expanding the local computational cluster seemed a promising idea, the emerging cloud computing paradigm caused us to investigate the utility of cloud computing for our application. Our assumption for the computational profile of cloud resources was the traditional view of IAAS cloud offerings: nodes with a variety of CPU and memory configurations, with commodity networking and the associated poor internode connectivity.

Here we discuss the process of process of determining the suitability of computations in the MG-RAST computational pipeline to the cloud computational paradigm. We begin by describing the design of the system, the MG-RAST computational pipeline, workload, and initial computational resources.

1.2 The MG-RAST system design

The MG-RAST system provides a web portal and a computational pipeline for the analysis of metagenomics datasets. Users submit sequence data sets via the MG-RAST portal, a web interface implemented as a series of Perl cgi scripts. This data is loaded into a Postgres database, along with information about which analytics should be performed and metadata about the dataset. The data is used, in turn, to submit jobs into the computational pipeline; in v2 the work was submitted through a traditional resource manager (Sun/Oracle Grid Engine); in v3 a locally developed workflow engine called AWE (Argonne Workflow Engine) is used to manage work execution. AWE is described below.

1.3 The MG-RAST computational pipeline (v2)

Once uploaded, the data exists in the form of standard sequence file formats accompanied by metadata (see the work of the GSC(7)). The pipeline consists of a series of perl scripts that execute various data transformations; each state is run as a job in the resource manager (Fig. 2).

The initial stage of the pipeline is basic quality control, detection of duplicate sequences, and calculation of some statistics. The sequence data is subsequently split into smaller

files and distributed to the computational cluster. Each sequent subset is run through BLASTX similarity searches, identifying matches of the nucleic acid sequence data to amino acid sequences in a large, nonredundant sequence database. While each sequence data subset is contains 100,000 characters of sequence data (or 100 kilobases), the nonredundant protein sequence library contains 1.7 gigabytes of amino acid data. This is a so-called pleasantly parallel problem, since the data can be subdivided into independent pieces with no interdependence.

Upon completion of the similarity searches, several derived data products are computed (e.g., reconstruction of microbial metabolism and a listing of species present in the data set) and subsequently loaded in the portal's database.

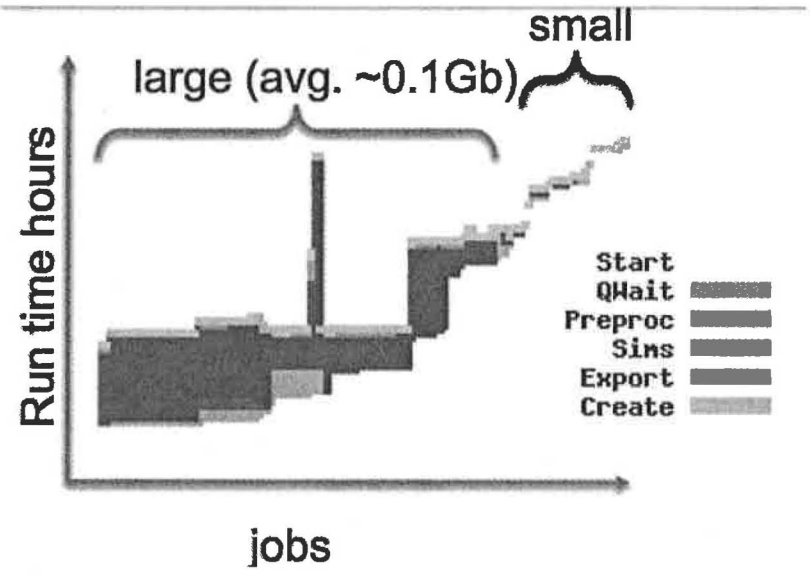

Figure 2: Time spent in different stages of processing for the MG-RAST v2 pipeline. The vast majority of time is taken up by similarity processing (purple). The almost constant amount of time spent in creating on disk representations (create) was corrected in a maintenance release of the v2 pipeline.

1.4 Local computational resources

The computational system that powers MG-RAST is a fairly standard Linux cluster managed by Sun Grid Engine, or SGE (8). It comprises 32 compute nodes in a variety of CPU and RAM configurations. Data is mounted on every node via NFS, with a standard user environment. An in-house software package was written to automate the running of the pipeline. It handles management of tasks in SGE, book keeping, and interactions with the MG-RAST web interface.

1.5 Discussion

MG-RAST's original architecture depended on a tightly coupled system environment. While this approach simplified development processes, it limited our computational operation to local resources that we controlled. This limitation increased the analysis

backlog as MG-RAST grew in popularity. In particular, we were unable to take advantage of other computational resources provided to the scientific community at large.

Procurement of dedicated hardware also poses difficulties. The lead times involved in procurement and deployment of hardware limit our ability to react to peaks in demand in a timely fashion.

2. Use of distributed heterogeneous resources

Because of a combination of the increasing demand for analysis via MG-RAST, as well as the quickly increasing data set sizes, dedicated computing resources will not provide sufficient capacity even for the short term, with considerably larger shortfalls in the medium and long terms. When tackling this problem, our first approach was to scale out the MG-RAST backend to existing shared resources at Argonne, followed closely with the use of cloud resources. This change posed some technical challenges because of differences in the general infrastructure provided on shared systems, as well as the switch to using distributed resources.

2.1 Technical challenges

Several high-level issues quickly presented themselves as we adapted the MG-RAST pipeline to the use of distributed resources. First and foremost, we realized that we had an orchestration problem; MG-RAST had previously depended heavily on the use of Sun Grid Engine for job execution reliability and prioritization, which was not in use on these shared resources. Analysis operations had a variety of system infrastructure requirements, some requiring large amounts of memory and local disk space or producing large quantities of output.

2.2 Analysis operations

An examination of the computational needs of analysis stages in the MG-RAST pipeline showed that they fell into a three broad categories: data-intensive applications, large-memory applications, and applications with long runtimes and pleasant parallelism. Table 1 describes each of the major stages in the pipeline in terms of the overall fraction of runtime, CPU intensity, memory requirements, and data requirements. By far the largest share of runtime, and hence the analysis bottleneck, came from similarity computations; it also had the simplest set of infrastructure requirements, so we distributed that stage of the pipeline first.

Table 1. Resources required for various stages of metagenome analysis in MG-RAST

Name	Fraction	CPU	Memory	Data Requirement
Quality Control	+	+	+	Full dataset
DNA/Peptide Clustering	+	+	+	Full dataset
Assembly	++	++	++++	Full dataset
Similarities	++++	++++	++	Subset
Export formatting	+	+	+	Subset
Database loading	+	+	+	Subset

2.3 Wide-area computational orchestration

We depended on Sun Grid Engine for analysis execution reliability and control. Initial inspection suggested that this function was simple enough to easily implement, so we implemented AWE to coordinate execution between multiple distributed systems. The similarity analysis stage was our initial target. This stage is implemented by using NCBI BLAST. It is a good candidate for distribution, both because it has a large resource consumption and because the computation uses a large, fixed database that changes infrequently. The only input to this stage is a small query sequence, easily transmitted from AWE (see Fig. 3).

Several other systems provide similar capabilities, including BOINC (9) and Condor (10). In our case, we chose to implement AWE in order to model work units in a format specifically suitable for genome sequence data. For example, because of the fixed metagenome database used by MG-RAST, request deduplication can be implemented if the AWE can recognize previously executed work units. Because of the relative simplicity of this software, we decided to build a purpose-built orchestration system, as opposed to adapting our workload to an off-the-shelf system.

AWE consists of a centralized set of python daemons that can communicate with clients via a RESTful interface. This approach is widely portable: clients need only to be able to perform HTTP requests to the server. The work queue, as well as results and statistics, is stored in a Postgres database. We have used Facebook's Tornado framework to build a lightweight and efficient set of front-tier web servers.

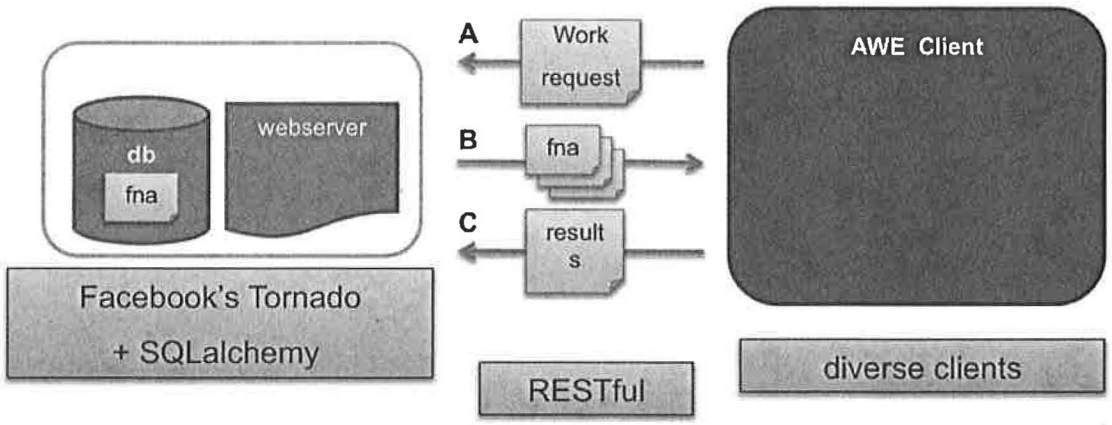

Figure 3: Schematic overview of the AWE design. Client and server use a RESTful API to communicate; the client obtains a lease on the analysis of a specific data set. Server and client side are implemented in Python with the server side using Tornado (CITE) and SQLalchemy.

Work units are stored in small chunks. When a client requests work, it includes a work unit size; this allows the client to size a request to the resources available. At this point the small chunks of work are aggregated into a larger unit, and each of these chunks is marked with a lease, including start time and duration. The client, as long as it is running, can extend this lease. If work units reach the end of the lease, they are freed for allocation to other client requests. This mechanism provides basic reliability in the face of unreliable remote clients. Each client can run one or more of these work unit cycles, depending on the resources available. On a current commodity system, these work units typically take 10–15 minutes to complete.

While the initial implementation was relatively straightforward, we found that we needed to get address a wider range of error conditions than we had previously encountered. Sun Grid Engine had handled many of these issues for us. This situation highlights one of the important lessons we learned in this process; most pipelines are still user-level software, and not system software. By implementing AWE and moving to resources we did not control, we began dealing with a wider range of error conditions. Moreover, these errors happened in a distributed environment, making debugging considerably more difficult. Similarly, debugging distributed performance problems required more sophisticated information gathering than when MG-RAST ran on a single cluster.

After becoming accustomed to dealing with performance issues, we were able to retune our database, modify our work management reliability scheme, and scale a single backend up to 500 compute nodes without issue. Further optimizations can be applied when they are needed; the architecture is designed to scale among multiple backend machines. However, since our current approach has been sufficiently scalable for the compute resources we have access to, we have not pursued this issue.

2.4 Confidentiality and data security in distributed platforms

Using a diverse set of heterogeneous computers might present security issues if access to an arbitrary subset of the data being analyzed is unacceptable to the end user. In our use case, access to subsets of DNA sequences without contextual metadata (see, e.g., GSC maintained metadata types (7, 11)) will render the data useless. The risk of accidentally granting access to anonymous DNA sequence data without any provenance information (as being distributed by the mechanisms we implemented) is deemed acceptable for the application we describe.

However, any existing sequence identifiers generated by sequencing equipment could also be rewritten to provide increase anonymity. In our application we strip out data protected by HIPPA before submitting sequence data to distributed resources.

2.5 Performance

Moving from an integrated local cluster environment to a more distributed type of work-load distribution has allowed us to utilize up to 500 nodes, whereas in the past we were restricted to much more limited locally available resources. We have been able to use available cycles from a number of machines including the DOE Magellan cloud testbed located at Argonne National Laboratory, Argonne's Laboratory Computing Resource Center computer, and Amazon's commercial EC2 offering.

One surprising result is the fact that because data volumes transferred after the initial deployment are small, even a DSL-type connection provides suitable throughout for our application model. Results of sequence comparison tend to be significantly larger than the input data sets; indeed, they frequently are10 times the size of the input sets. Here using our domain knowledge, we have chosen to change the report format for BLAST to produce results with a significantly reduced on-disk and data transfer footprint.

3. Guidelines

In adapting MG-RAST to cloud and ad hoc computational resources, we have developed a number of guidelines for determining how best to utilize such resources.

3.1 Characterize applications

Applications have a wide range of properties. Serial codes are well suited to clouds, whereas parallel codes are better suited to local clusters. Some applications are highly I/O intensive, limiting their performance on cloud resources, whereas others are not. Each of these characteristics determines how effectively remote resources can be used and whether their use for the task is worthwhile. Similarly, cloud resources have prices associated with access; this information can be used to develop a cost model for analysis that combines the price performance of resources for the throughput achieved for a given application on that resource.

3.2 Consider portability

Without appropriate care, applications can embed assumptions about the software configuration of computational resources. These assumptions limit the portability of applications, restricting the ability to move computations from one facility to another.

When resources become available, it is often advantageous to be able to make use of them quickly.

3.3 Prepare for scalability

The workload for a local computational resource is often very regular; a moderate number of resources will be available overall. In composite environments including both local and remote resources, the aggregate availability of resources can quickly change. These spikes can cause scalability problems. Scalability issues should be considered during the design and tested with synthetic workloads prior to deployment, if possible.

3.4 Develop effective logging approaches

Many pipeline developers are familiar with application development in a localized setting. Cloud software is system software running in a distributed context. This means that telemetry, logging, and other debugging mechanisms are critical; information collected must be comprehensive, to allow post hoc debugging of system issues. The collection of this data can pose scalability problems, so an organized analysis of logging as well as the determination of the log messages required to debug common issues should both be performed.

4. Conclusion

The explosion of sequencing capacity in the bioinformatics community has cause a large and ever-growing need for computational cycles. The bioinformatics community as a whole, as well as pipeline operators in particular, need to be prepared to make use of distributed resources in order to be able to satisfy the demand for data analysis.

In this paper, we have presented our experiences in adapting MG-RAST to make use of distributed resources for its computation. This process was not a simple one; major alterations to our runtime infrastructure, operational models, and debugging processes were required. While the cost of these changes was high, we feel that these adaptations were required in order to scale the current and future dataset sizes and analysis demands.

On-demand architectures for computation have enabled us to perform analysis for larger data sets in larger quantities; however, this alone is not a panacea. Clouds have enabled us to determine the market costs of analysis, but these quantified costs have shown us that we cannot financially afford to continue performing our analysis with the current generation of algorithms, even taking Moore's law into account. More efficient analysis algorithms will be needed soon.

Acknowledgments

This work was supported by the U.S. Dept. of Energy under Contract DE-AC02-06CH11357.

Bibliography

1. Committee on Metagenomics: Challenges and Functional Applications NRC. The New Science of Metagenomics: Revealing the Secrets of Our Microbial Planet2007.

2. Tyson GW, Chapman J, Hugenholtz P, Allen EE, Ram RJ, Richardson PM, et al. Community structure and metabolism through reconstruction of microbial genomes from the environment. Nature. 2004;428(6978):37-43.
3. Venter JC, Remington K, Heidelberg JF, Halpern AL, Rusch D, Eisen JA, et al. Environmental genome shotgun sequencing of the Sargasso Sea. Science. 2004;304(5667):66-74.
4. Kent WJ. BLAT--the BLAST-like alignment tool. Genome research. 2002;12(4):656-64. PMCID: 187518.
5. Langmead B, Trapnell C, Pop M, Salzberg SL. Ultrafast and memory-efficient alignment of short DNA sequences to the human genome. Genome biology. 2009;10(3):R25. PMCID: 2690996.
6. Li R, Yu C, Li Y, Lam TW, Yiu SM, Kristiansen K, et al. SOAP2: an improved ultrafast tool for short read alignment. Bioinformatics. 2009;25(15):1966-7.
7. Kottmann R, Gray T, Murphy S, Kagan L, Kravitz S, Lombardot T, et al. A standard MIGS/MIMS compliant XML Schema: toward the development of the Genomic Contextual Data Markup Language (GCDML). Omics. 2008;12(2):115-21.
8. gridengine -- Project home. 2007 [updated 2007; cited]; Available from: http://gridengine.sunsource.net/.
9. BOINC. 2011 [updated 2011; cited]; Available from: http://boinc.berkeley.edu/wiki/Main_Page.
10. Tannenbaum T, Wright D, Miller K, Livny M. , "Condor - A Distributed Job Scheduler",. In: Sterling T, editor. Beowulf Cluster Computing with Linux: The MIT Press; 2002.
11. Field D, Garrity G, Gray T, Morrison N, Selengut J, Sterk P, et al. The minimum information about a genome sequence (MIGS) specification. Nature biotechnology. 2008;26(5):541-7. PMCID: 2409278.

To be removed before publication:

The submitted manuscript has been created by the University of Chicago as Operator of Argonne National Laboratory (``Argonne'') under Contract DE-AC02-06CH11357 with the U.S. Department of Energy. The U.S. Government retains for itself, and others acting on its behalf, a paid-up, nonexclusive, irrevocable worldwide license in said article to reproduce, prepare derivative works, distribute copies to the public, and perform publicly and display publicly, by or on behalf of the Government.