

Enabling Automatic Data Analysis in Bioinformatics Core Facilities

A high-throughput sequencing pipeline

ROMAN VALLS GUIMERÀ



**KTH Computer Science
and Communication**

Master of Science Thesis
Stockholm, Sweden 2012

Enabling Automatic Data Analysis in Bioinformatics Core Facilities

A high-throughput sequencing pipeline

R O M A N V A L L S G U I M E R À

DD225X, Master's Thesis in Biomedical Engineering (30 ECTS credits)
Master Programme in Computational and Systems Biology 120 credits
Royal Institute of Technology year 2012
Supervisor at CSC was Lars Arvestad
Examiner was Jens Lagergren

TRITA-CSC-E 2012:020
ISRN-KTH/CSC/E--12/020--SE
ISSN-1653-5715

Royal Institute of Technology
School of Computer Science and Communication

KTH CSC
SE-100 44 Stockholm, Sweden

URL: www.kth.se/csc

Abstract

A genomic sequencing experiment, from sample sequencing to data analysis requires a system that can be easily adapted to researchers' needs. Currently, repetitive and error-prone steps need manual attention from specialists. This sub-optimal situation reduces the time available for personnel to work on more interesting problems.

All of this must be done while a production system is running, so parallel development, testing and production environments are needed to provide better reliability. Furthermore, software development practices such as version control and continuous integration systems must be put in place to ensure source code quality and improved collaboration between bioinformaticians.

The goal of the thesis is to re-use and extend a modular software to efficiently process the growing amount of sequencing data. The ideal outcome is a system where bioinformaticians and biologists can focus and collaborate with state of the art analysis tools without worrying about repetitive tasks that offer no added value to their research.

Here we present an open source integrated solution that solves/remediates the problems outlined above by using sound software engineering methods.

Contents

1	A bioinformatics core facility	3
1.1	Introduction	3
1.2	Background	4
1.3	Systems infrastructure	5
1.3.1	HPC environments	6
1.3.2	Cloud computing	7
1.4	Pipeline software blueprints	8
1.5	Pipeline software reality	9
1.6	Laboratory Information Management Systems	11
2	Methods	14
2.1	Tools for bioinformatics developers	14
2.1.1	Distributed version control systems	14
2.1.2	Isolated working environments	15
2.1.3	Module system considered harmful	17
2.1.4	Continuous integration system	20
2.2	Contributing code	20
3	Pipeline integration	22
3.1	Storage	22
3.2	Processing	26
3.3	Network	27
3.3.1	Lsyncd	28
3.4	DRMAA connector	28
3.5	Blue Collar Bioinformatics pipeline	30
3.6	How does bcbio-nextgen work ?	31
3.7	Upstream pre-processing	31
3.8	Downstream analysis	32
3.8.1	Samplesheet	34
3.9	Galaxy	34
3.9.1	Galaxy deployment on HPC	34
3.9.2	Tunneled Galaxy access	35
3.9.3	Apache	38

3.10 Empirical tests	38
4 Final remarks	40
4.1 Conclusions	40
4.2 Future work	41
4.3 Acknowledgements	41
4.4 Appendix	42
Bibliography	44

CONTENTS

Outline of the report

The first chapter consists of a non-technical overview on the field of scientific workflows, bioinformatics software, laboratory information management systems and its associated operational issues.

The second chapter consists of a brief explanation on how to work on open source scientific environments and get acquainted with the bioinformatics community in the best way.

The third chapter of the report is more technical. The inner working details of the integration between different pipeline components are described and discussed.

Nomenclature

AMQP Advanced Message Queuing Protocol

API Applications Programming Interface

CDE Code Data Environment

CSV Comma Separated Values

DAG Directed Acyclic Graph

DRMAA Distributed Resource Management Application Api

DSL Domain Specific Language

DVCS Distributed Version Control System

GIL Global Interpreter Lock

GUI Guided User Interface

HPC High Performance Computing

HTTP HyperText Transfer Protocol

LIMS Laboratory Information Management System

NDA Non disclosure agreement

REST REpresentational State Transfer

SGE Sun Grid Engine

SLURM Simple Linux Utility for Resource Management

TOS Terms Of Service

WSDL Web Service Definition Language

YAML YAML Ain't Markup Language

Chapter 1

A bioinformatics core facility

1.1 Introduction

This thesis originates from the need to process vast amounts of genomic data [PBK10]. Several genomics facilities around the world are experiencing similar data handling difficulties.

In a genomics sequencing facility, there are instruments that take biological samples from diverse organisms as input and generate the computer representation of their genomic sequences as output. The underlying processes on how to perform the sequencing are purely platform-specific [Gle11]. The focus in this thesis will be on the output sequences, where the *in silico* data management and its handling intricacies begin [RS09], as opposed to *upstream* wetlab sample processing.

Since the logic to process data manually would require a substantial human effort, it is desirable to put an automated yet flexible system in place that can serve bioinformatics research. In particular, repetitive and error-prone steps that need manual attention from specialists should be avoided as much as possible by software means. A system that addresses the aforementioned problems is a bioinformatics *pipeline* or *workflow*.

The concept of a genomics pipeline comes into play when a bioinformatics researcher starts to use more than one bioinformatics tool. One wants to connect two or more software applications to assess qualities, compare results, convert formats, visualize data and perform many other operations in a coherent way. Therefore a pipeline (or workflow) consists of different programs “glued” [vR98] together, performing tasks of varying complexity.

From a human resources perspective [KWV11], it is easy to see how failing to achieve a degree of automation on repetitive processes reduces the time available for qualified personnel to work on research problems [LR09]. As an example, in some contexts, bioinformatics research is performed by individuals writing private *ad hoc* scripts to quickly answer specific research questions [DB09] as opposed to using semi-automated parametrized workflows [AAG10, EHT10], maintained by an open source community of researchers with similar needs to answer particular research questions.

As a result, automation of commonly performed tasks, which in turn form complex workflows, frees up researchers from tedious command-line sessions, non-reusable script writing, and general time-consuming software “firefighting” [LRRS09].

The overall assignment for this thesis is to automate bioinformatic analysis, adopt and disseminate best practice methods (chapter 2) on software development and integrate or fill the existing gap between computing infrastructure and bioinformatic applications. The approach taken is to adapt an already existing open source high-throughput genomics pipeline. That is, by extending and fixing different middleware software components (chapter 3), a coherent, reusable and completely open source pipeline is assembled.

Firstly, gathering metrics and exploring different design options in terms of data storage, process time and transfer speed is crucial to decide what is the best option for both current and future facility needs. In particular, predicting future technology updates and how they could affect the underlying data infrastructure is highly useful to foresee system design limitations.

Secondly, a source code and data collaboration strategy should be established so that development can survive on a fast-changing running system. The requirements can change overnight, which may require generalization. In addition, a team of researchers demand fast delivery of results from their experiments, imposing further pressure on software design decisions. As a result, extra care must be taken when writing, testing and deploying software. Such a strategy and best practices will be detailed.

Lastly, empirical tests will be presented. Different in-house research tools that solve specific needs will be integrated in the framework presented, using the techniques developed in this thesis.

1.2 Background

When trying to understand a complex system it is always desirable to have a birds-eye view on how the different pieces hold together, as illustrated by the abstract stack in figure 1.1. In the context of a bioinformatics core facility, following a bottom-up approach, we first find hardware. Those are the computer resources and its architecture, the *infrastructure* that system administrators manage and users operate. Following further up in the stack, different bioinformatic tools, *pipeline software* and their inner algorithms run on top of the infrastructure.

Finally, closer to human beings, there are the so-called laboratory information management systems, *LIMS*, which allow lab scientists to interact with data experiments and their attributes.

Obviously this is a simplified view, but that is enough to describe how the relation between software components look like and where this project unfolds.

1.3. SYSTEMS INFRASTRUCTURE

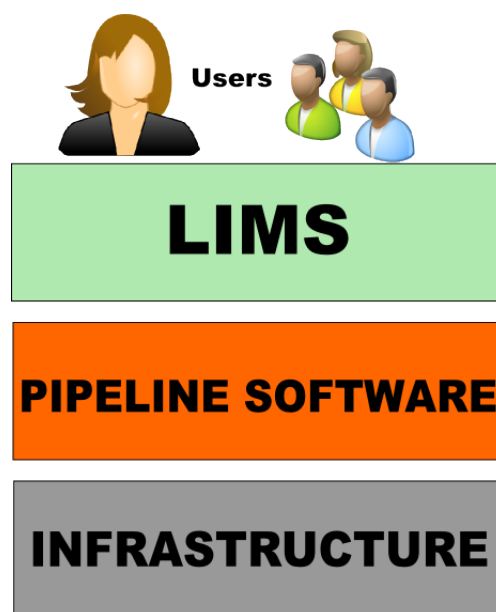


Figure 1.1. Abstract stack of the different high level domains in a bioinformatics core facility.

1.3 Systems infrastructure

As a result of the rapid developments on the genomics sequencing field, instrument manufacturers are reducing the cost per genome sequenced, as illustrated in figure 1.2. The immediate consequence of the price dropping is an increase of data production [Cal11]. With no evidence of the so-called “data deluge” slowing down in the coming years, it is easy to foresee an increasing demand on storage and processing needs [PBK10]. Finding a scalable way to store and process all generated data in the most efficient, accessible, metadata-rich, secure and transparent way is not a trivial endeavour.

Big data [Doc08, MK12] systems have been invented to deal with the inability of traditional approaches to cope with hundreds of terabytes of data. Big data can be thought as a way to store and analyze extremely large amounts of data (today on the petabyte range) in the most convenient way, often by using new data processing paradigms such as *MapReduce* [DG08].

But perhaps more importantly, from a systems perspective, virtualization [BDF⁺03] and cloud computing sketch a different future, evolving from today’s supercomputing resources (section 1.3.1), to the so-called cloud as briefed in section 1.3.2.

Not only the data itself poses a big challenge for the systems, but so does the *metadata* that comes along with it [RSBM⁺10]. In short, *metadata* allows data scientists and biologists to have uniformly structured “book keeping” and extra details about the data [RSBM⁺10]. Systematizing routine data handling operations with the help of robust *metadata* implementation and accompanying consistency

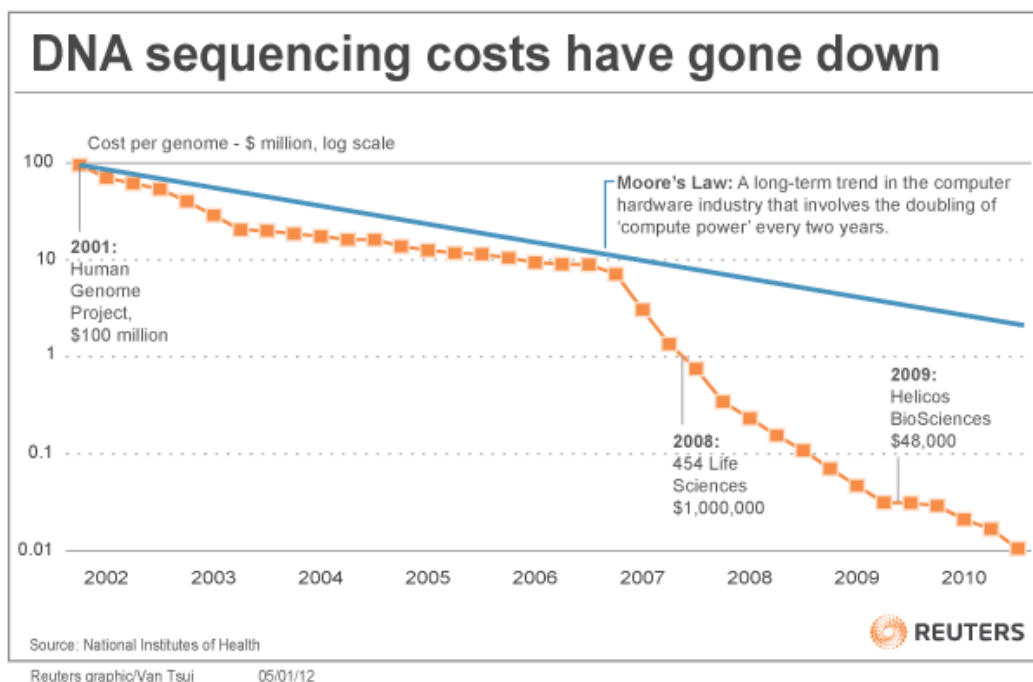


Figure 1.2. Costs per genome over time.

checks, help with reducing human errors such as accidentally deleting data from ongoing experiments.

Again, systematizing everything, even the infrastructure itself increases reliability when reproducing experiments. With the advent of cloud computing [Ste10], virtual clusters can be spawned with few lines of code. On the other hand, not all computing environments have had the time or resources to adapt to private cloud computing and further explore their advantages and drawbacks in real settings.

1.3.1 HPC environments

Traditionally, a high performance computing cluster (HPC) is implemented as several computers (nodes) connected together via a fast network. In addition, a shared filesystem that can be accessed by all nodes conforms the basic idea of a so-called “Beowulf cluster” [Wik11a], widely deployed both in academia and industry.

Since a supercomputer is meant to be shared by several users, it typically runs a queuing or “batch” system which schedules and prioritizes how different tasks should be run. Software installed in the system is also shared across nodes, so several instances of the same software can run on different nodes.

In this setting, we can work with the system as if it was a regular computer, except that there are two important system-dependent details that are not standard across different cluster installations:

1.3. SYSTEMS INFRASTRUCTURE

1. One has to interact with the batch or queue system being used.
2. Users must know about how software management works in a particular HPC environment.

The first HPC system-dependent detail, the batch system, can be abstracted by an extra software layer that encapsulates or “masks” platform incompatibilities. In other words, similar actions performed by different batch systems (SGE, LSF, SLURM, etc...) can be generalized, providing a common abstract way to talk with the different systems. Details on how this is done are explained on chapter 3.

The second issue, software management, poses a challenge to generality since there is no standard across universities and systems on how to install and update software. Nevertheless, a possible solution and future recommendations are detailed in chapter 2.

1.3.2 Cloud computing

Despite the advances on bioinformatics tools, analyzing DNA sequences is a skill that requires knowledge about different tool’s pros and cons. If on top of that inherently complex processing we add infrastructure-specific details explained previously in section 1.3.1, data analysis becomes accessible only to few highly skilled professionals. Nowadays, one of the most pressing issues for researchers is having access to infrastructure that works and helps them advancing on their research [Dee09, TD10] in a seamless and easy to approach way [ABC⁺11], as opposed to paying attention to system specific implementation details.

Cloud computing gives users full control over (virtual) compute resources, by using virtualization technologies [BDF⁺03]. For instance, a researcher can form virtual HPC environments to execute experiments and later on, share the entire setting with fellow researchers [ABC⁺10]. However, without proper automation, setting up virtual clusters becomes an arduous task. As any system administrator knows, it requires a considerable amount of effort to configure, install and fine tune a HPC system. Therefore, deployment and provisioning recipes or scripts are crucial for the cloud computing model to succeed.

Once the basic infrastructure is up and running, orchestrating the different bioinformatics software components [Cha11a] that run on top is still a hot topic. Today there are existing implementations that keep researchers away from steep learning curves by using pre-made software installations in the form of domain-specific cloud computing machine images [ABT⁺11, Cha11a]. In essence, this machine image is a virtual representation of a hard disk drive containing all the software and data preinstalled. Many “instances” or copies can be made of this image, as many as the different experiments one might perform on a computer.

Unfortunately, several arguments against cloud computing are still driving institutions away from this model. Monetary cost being a frequent concern has been already explored by biomedical researchers at Harvard Medical School [FPG⁺11]. Their experiment resulted in costs incurring less than \$400, running a 304 CPU

cluster on Amazon’s cloud offering [Gar07] to process data during 10 hours, as illustrated in figure 1.3. Afterwards, the whole cluster was terminated, in line with the so called *utility computing* [BYV⁺09], effectively cutting costs down when the computing resources were not being used.

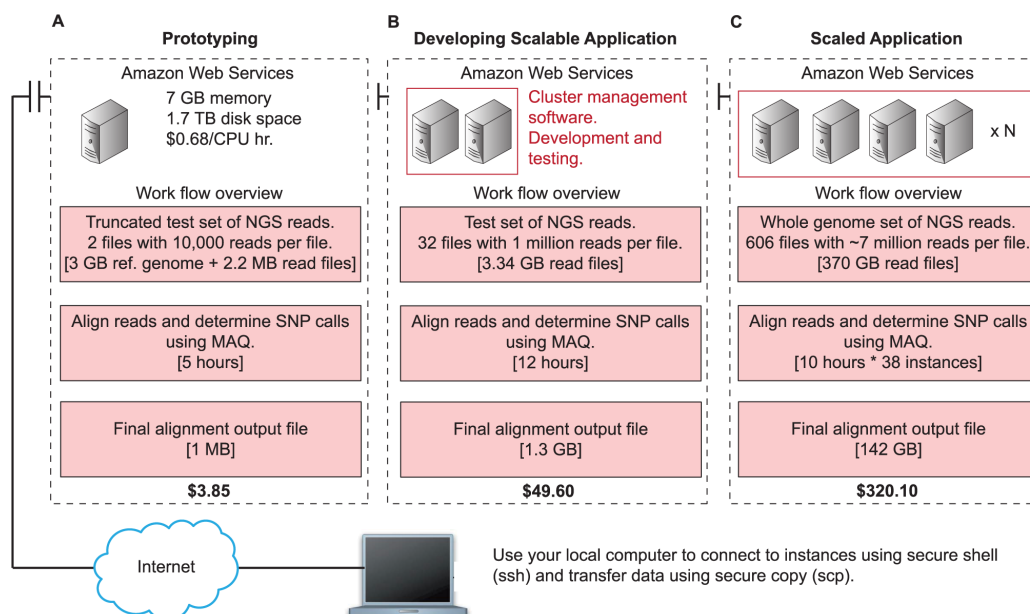


Figure 1.3. Step-wise framework for creating a scalable NGS computing application and its associated costs on Amazon AWS, as shown in [FPG⁺11].

Another recurrent issue raised by many experts is related to today’s network bottlenecks and the time it takes to send big data up in the cloud and back. Fortunately, this can be alleviated by sending physical media over traditional regular mail, at least until better compression methods or more generous network pipes are put in place. Finally, funding, privacy concerns and strict laws regulating data have to be accounted for [Cat10, GSMG11, FSC⁺09, BSB04] since, for instance, approval (consent forms) from patients and liability varies over different studies, experiments and countries.

Since existing resources, while writing this thesis, were traditional HPC clusters, focus and limitations fall in this area, but preparation for cloud computing is always kept under consideration for the future.

1.4 Pipeline software blueprints

In the omics field, there are different tools that aim to solve the problem of extracting valuable scientific insights from large amounts of data on high-throughput genomics experiments. There is software being actively developed at the moment, either by open source individuals or proprietary independent software vendors.

1.5. PIPELINE SOFTWARE REALITY

A good blueprint for a pipeline should meet common design criteria [BCH⁺10]:

1. Open: Any developer must be able to change any aspect of the software. This is not always possible to a full extent with proprietary software.
2. Automated: Most of the operation time, it should be run completely unattended with minimal human intervention.
3. Extensible: Must have an acceptable level of abstraction so that one can accomodate future changes easily.
4. Restartable: One should be able to restart the pipeline in case of unexpected failure or change of initial parameters.
5. Reproducible [Pen11, Gen05, Mer10a, Ioa05]: The software should be able to reproduce previous results when re-running or restarting it.
6. Easy to use: Users should be able to interact with the software in an intuitive way.
7. Deployable: Should not require a large upfront investment on installation and configuration.
8. Efficient: From data production to delivery, processing should be as fast as possible without falling into the trap of premature optimization¹. In other words, human engineering time is generally more expensive than CPU time.

Even if all the blueprints exposed should be given full care and attention when working on improving them, the reality is quite different. Different laboratories have different priorities and consequently they would implement and focus more on a subset of those blueprints. My personal take on this regard is that science in general *must* be open, and that is why the corresponding blueprint is listed first. All other facets stem from openness in that they can be improved over time provided enough diligence and effort.

1.5 Pipeline software reality

Despite ongoing efforts, omics pipelines are still a relatively young topic in computational biology and big data science. Therefore, software that satisfies the above points in the best way is still hard to find. Moreover, pipelines blend knowledge from many different disciplines. That is, ranging from system administration, software design and usability to biotechnology and computational biology. In other words, a high quality pipeline should be one orchestrating many of those facets gracefully.

Today it is reasonable to say that none of the pipelines implement all features required by high throughput bioinformatic facilities. To make things worse, many

¹“(...)premature optimization is the root of all evil” – Donald Knuth

of those developments lack enough user base to be viable as production solutions in the future. As will be explained in chapter 2, lowering the barrier to entry on deployment for developer and user communities greatly helps when building general, reusable and interoperable software [Ste96].

Reimplementing already existing functionality hinders interoperability between bioinformatics tools, perpetuating the already brittle compatibility in the bioinformatics formats and standards ecosystem. As an example, tools like GATK, Picard and snpEff introduced changes that broke file format and commandline compatibility with previous versions while this thesis was being written, making software and dataset maintenance an unnecessary administrative hurdle.

Purpose-specific pipelines may help to solve problems on their domain, but they would be more useful for the field in general if they were packaged as interoperable programs or web services. There are also solutions such as Conveyor [LGG11] that put an extra effort on the GUI, disregarding basic functionality such as support for BAM and SAM formats [LHW⁺09], almost de facto file format standards for genome alignment. Last but not least, pipelines designed to be run assuming huge computational resources impose an extra cost on setup, deployment and maintenance. One would say those need to “scale down”, simplify and put an extra effort on software packaging to consolidate their deployment.

Pipelines try to be holistic and handle different aspects of sequencing in a simple way. Unfortunately, some are too generic and lack common use cases [CM11], many others focus on very specific problems [GTBK11] and therefore cannot be easily reused for other tasks. Having discarded pipelines that don’t adhere to the standards outlined in 1.4 enumeration, there are at least two frameworks that can fulfill the requirements: Taverna and Galaxy.

Taverna [HWS⁺06] provides a workflow management architecture based on standard webservices (WSDL). Over years of development and community support, Taverna has developed a solid repository of ready-made computational biology workflows ranging from simple file format conversions to more complex SNP calling or pathway analysis [LOSK08]. Taverna is built around good software design principles, enforcing the idea that a non tech-savvy user should be able to graphically connect components (tools) while retaining programmatic access to components via an API for more experienced users.

Despite its extensive documentation, workflow repositories and educational resources, the learning curve and integration time [RWC⁺10] for this system shouldn’t be ignored. Taverna’s “workbench” computational demands might not allow it to run on all platforms and devices, in contrast, other web-based systems only require a web browser, much more common and lightweight than a full JVM stack.

On the other hand, Galaxy [GNTG10] workflow system presents a web interface to the user that can be rendered on most web browsers, totally ubiquitous in today’s personal computing. On top of that, the thriving community that surrounds Galaxy and several public servers being deployed by different universities to expose their internal tools are good incentives to adopt this platform. Furthermore, its integration with HPC systems, its cloud computing approach using CloudMan [ABC⁺11]

1.6. LABORATORY INFORMATION MANAGEMENT SYSTEMS

suggest that special care has been taken on deployment strategies and therefore, generality. Consequently, Galaxy not only seems well suited but initiatives such as GenomeSpace [Gen11b] promise to deliver future interoperability between tools such as Taverna [Kar11] or GenePattern [RLG⁺06].

Unfortunately, Galaxy does not yet deliver an automatic analysis pipeline, its API access is rather limited compared to Taverna's, but it is quickly improving [Gui11b]. Furthermore, compatibility with SLURM batch management system, crucial for dispatching actual work to the cluster, is not currently supported. Therefore, the focus of this thesis is to solve these shortcomings within the open source bioinformatics community [PLS⁺11] by extending bcbio-nextgen pipeline 3.6 and integrating it with Galaxy.

1.6 Laboratory Information Management Systems

A LIMS is supposed to support many lab operations such as sample tracking, instrument monitoring and metadata exchange with other systems. In some sense, LIMS features are very dependant on the actual institutions and can vary enormously between organizations [Bio11b, Wik11c] and their assumptions within their particular working environments. Today's high throughput biology does require structure and automation when keeping track of concentrations, samples, reagent batches or abstract workflows, to name a few attributes.

Given these premises, designing a generalized LIMS is not a trivial task [WSP⁺07]. Traditionally, the core of a LIMS is a database where all the data elements and their relations are pre-defined during the design phase of the system. Since storing, organizing and fetching data in a LIMS is not considered an active research field, most of the times neglected in favour of algorithmically complex assemblers and/or visualization tools (both arguably very different topics or classes of software) [WSP⁺07], there's not much research and/or robust tools in this area.

Commercial LIMS such as GeospizaTM [Lim11] have succeeded to some extent in several genomics facilities. For example, Geospiza uses the concept of SQL query API where the administrator can define arbitrary queries and encapsulate those as a query that can be triggered by REST [Tom04], a standard programmatic way to interact with the web. Unfortunately Geospiza is not flexible enough for all settings. Simple operations such as gathering metadata from sequencers² are not currently supported as an API offering. Moreover, instrument run finalization is not handled automatically, violating the principle of automation defined earlier in 1.4. Besides this, being a closed source vendor, those limitations cannot be addressed by independent software developers³, once again it does not align well with our requisites.

²Retrieving an Illumina's samplesheet programmatically

³The (relatively long) time it takes to release new updates does not help on getting limitations fixed quickly either.

Another prominent commercial LIMS example is GenoLogics™ [Gen11a], which offers a clearly documented API access via REST. Unfortunately the rest of the software suite remains as proprietary technology, preventing further custom feature additions by core staff.

As a result of the intrinsic proprietary LIMS, some open source alternatives, such as Sanger’s “sequencescape” [WTS10] are appearing. As an example, Brad Chapman’s ngLIMS [Cha11b] integrates a full sample tracking workflow for Illumina™ inside Galaxy, as illustrated in figure 1.4:

The screenshot shows the Galaxy / SciLifeLab.se interface. On the left is a sidebar with navigation links: Samples (Define samples and services, Submit samples as a project, View projects, Sequencing results), Sequencing (Queues, Runs, Plots). The main area is titled 'Sample Information' and contains a form for adding a new sample. The form fields are: Name (ROa2), Description (Spruce genome), Genome Build (Human Feb. 2009 (GRCh37/hg19) (hg19)), Analysis (Standard), Multiplexed (checkbox), and Services needed (Library construction, Library validation, Next gen sequencing). Below the form is a table titled 'Current samples' with columns Name, Description, and Details. The table contains one row: ROa1, BC45521X, Services: Library construction, Library validation, Next gen sequencing; Sequencing: 100 paired. A note at the bottom says 'Click on a sample name for expanded information.'

Figure 1.4. Galaxy’s ngLIMS interface

As an example of well designed human-computer interaction (see 1.4 in usability), if the sample is multiplexed, it allows to drag and drop the different barcodes on the samples directly on the web interface illustrated in figure 1.5. Unfortunately, an API for ngLIMS is missing at this point.

Another interesting option is slog [Kra11], developed by Per Kraulis and freely available. Slog is a simple lab management system on top of the well established NoSQL [Sto10] database CouchDB [Fou11]. This particular approach using schema-less databases confers extra flexibility in the ever-changing lab protocols, workflows and instruments. Unfortunately its current status is still in alpha stage.

Last but not least, in some cases LIMS have to be aware of downstream analysis and dataset management. For this reason, an iRODS-aware system [RMH⁺10] is of high importance for provenance and more automated data management in the

1.6. LABORATORY INFORMATION MANAGEMENT SYSTEMS

Galaxy / SciLifeLab.se

Analyze Data Workflow Shared Data Lab Visualization Admin Help User

Samples

- Define samples and services
- Submit samples as a project
- View projects
- Sequencing results

Sequencing

- Queues
- Buns
- Plots

Sample Information

Preparation details

Barcodes

Summary

Barcodes

Illumina Custom

1 : ATCACG
2 : CGATGT
3 : TTAGGC
4 : TGACCA
5 : ACAGTG
6 : GCCAAT
7 : CAGATC
8 : ACTTGA
9 : GATCAG
10 : TAGCTT
11 : GGCTAC
12 : CTTGTA
unmatched : NNNNNN

Multiplex details

Click to edit name

6 : GCCAAT

Add new sub-sample

Current samples

Name	Description	Details
RQa1	BC45521X	Services: Library construction, Library validation, Next gen sequencing; Sequencing: 100 paired

Click on a sample name for expanded information.

Figure 1.5. ngLIMS with Javascript drag and drop support for barcodes

future (see 4.1). iRODS, which stands for “integrated Rule-Oriented Data System”, defines a common rule language to automate data and metadata provenance in a robust and standardized way. As an example, the Wellcome Trust Sanger Institute, a pioneering scientific facility, is using such a system in production to cope with petabytes of data [CCQ⁺11].

To sum up, designing and implementing a LIMS requires a considerable effort and lies out of the scope of this thesis. Nevertheless, since the pipeline has to interoperate with a LIMS when it comes to metadata handling and monitoring, learning about LIMS current status and how are they built is essential if one wants to understand how to exchange data in the best way.

Chapter 2

Methods

2.1 Tools for bioinformatics developers

In any mature and modern software development project there should be a surrounding set of tools to assist programming and management of source code [DB09]. Within a multidisciplinary working environment such as a life sciences facility, not all peers may be aware of the “tools of the trade” for computer engineering.

Despite the relative young nature of bioinformatics, several centers are starting to realise that solid software engineering practices still apply [RPP⁺11]. For that reason, this chapter will elaborate on how different tools and methods were used to work in a stable, organized yet highly collaborative [PLS⁺11] environment.

2.1.1 Distributed version control systems

When working with software it is always desirable to keep a log of changes, in a similar way a scientist keeps track of experiments in an electronic lab notebook.

In source code, this tracking is essential if one wants to collaborate concurrently with an open source community. In particular, the code which is based on this thesis is derived from an existing public source code repository. As opposed to the traditional central source code repositories, distributed version control systems (DVCS's) ease the access to source code by *forking* or making personal remote copies of different repositories. To put it simply, using a DVCS essentially liberalizes sharing of source code by letting peers control software instead of managers [O'S09].

The foundation of public DVCS repositories such as GitHub and BitBucket in 2008 added social features to code sharing, allowing developers to easily track each other's advances, issues and contribute fixes to the software. Different infographics show how different open source communities work together such as [HMRH11]. For example, this master thesis project's timeline can be seen in figure 2.1, where many developers work in parallel and merge code with each other using the concept of *branches* to improve the software as a whole.

Since Git DVCS allows for virtually unlimited branching, it is good to set some conventions on which branches are stable before sharing code. Therefore, git-flow

2.1. TOOLS FOR BIOINFORMATICS DEVELOPERS

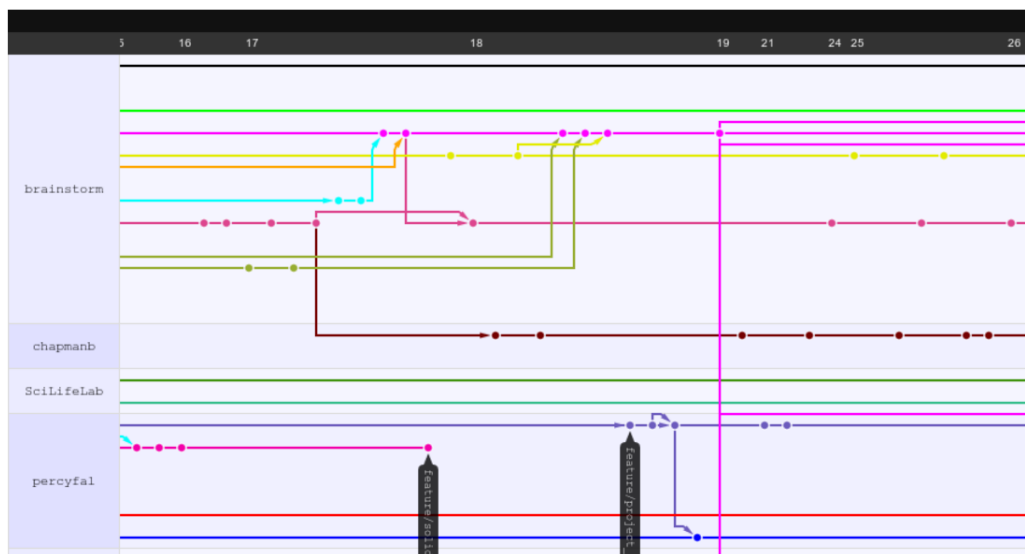


Figure 2.1. GitHub branch tree timeline based on our real case at Science For Life Laboratory. Different people contribute source code to the same project, yet, they have their own personal copies of the project. Eventually, personal changes are shared between peers by merging code.

[Dri11] provides such conventions and automates part of the process of setting up experimental ramifications of code by standardizing the semantics of different branches. To put it simply, the *master* branch constitutes the “production quality” code, the one that ultimately runs on the system. In contrast, the *develop* branch, is used for changing and experimental code that regularly gets merged back into master when it’s considered stable. Similarly, *feature* branches are used, as the name implies, to develop new features that get merged incrementally back into develop. Lastly, *release* branches mark milestones while *hotfix* branches provide quick last-minute production fixes. Therefore using this scheme, software is kept organized by a common convention and peers know what other developers are working on at any time, as shown in figure 2.2.

2.1.2 Isolated working environments

When working in a shared HPC environment, there are often limitations when it comes to software management and versioning. For example, different HPC users may have completely opposite needs: A developer wants latest versions for all libraries and programs, while a researcher might prefer stability while running her ongoing experiments.

Thus, it is important for the users to decouple from the specifics of a certain computing environment and be allowed to switch the “workbench” as desired, without infrastructural or policy constraints.

By the technical reasoning exposed in 2.1.3, virtual python environments [Gui]

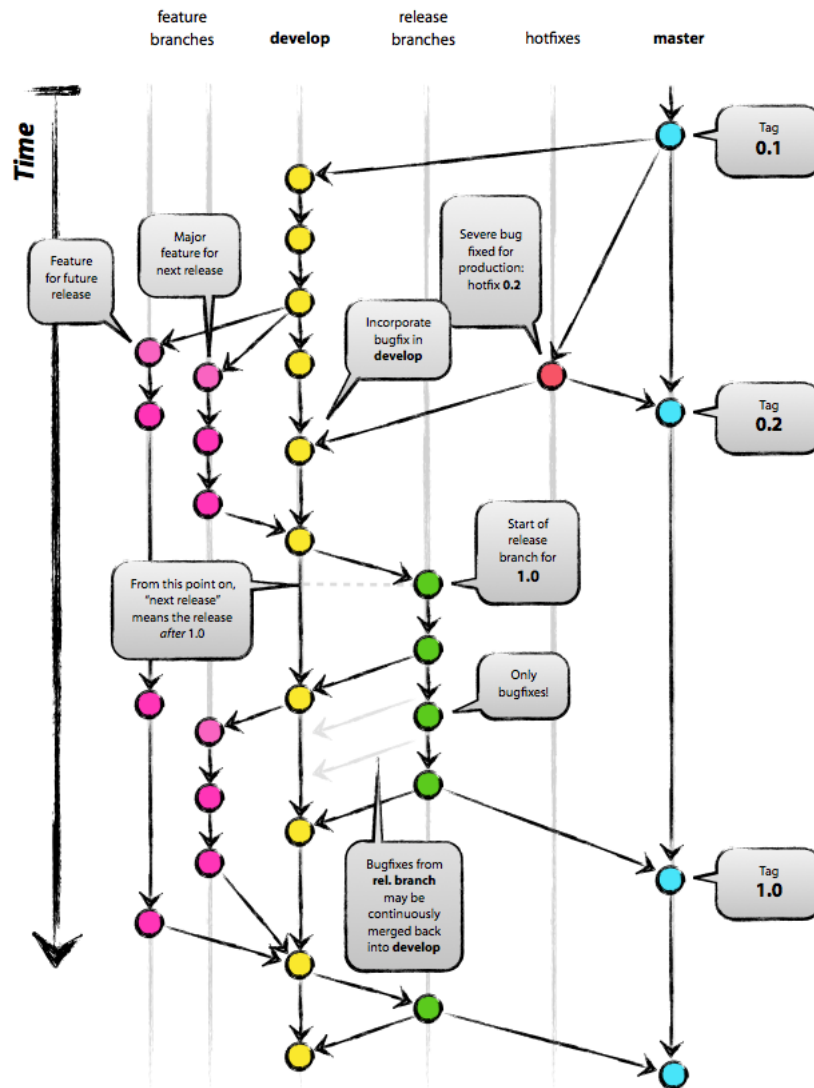


Figure 2.2. GitFlow branch tree representation where master is stable code and develop where new (unstable) features are introduced.

are used. A virtual Python environment allows any user to install Python modules and programs with independence from those already installed in the base system. Therefore, new software can be downloaded and deployed quickly, without intervention from the system administrators.

Each virtual environment is mapped one-to-one to repository branches to avoid confusions during deployment. For example, when working in a local Python virtual environment called “master”, that environment would correspond to the repository branch (and the versioned code therein) by the name “master” in the DVCS.

Additionally, checking out the repositories automatically on a daily basis ensures

2.1. TOOLS FOR BIOINFORMATICS DEVELOPERS

good version trackability, which helps when answering to the important question “was version X running in production when performing the experiment Y?”, instead of just relying on application logs.

Lastly, log files containing the versions of the different software running inside the virtual environment is kept for provenance. In particular, versions used by the module system, introduced in the next section.

2.1.3 Module system considered harmful

Dealing with software package management can be a daunting task, even for experienced system administrators. From the long forgotten *Graft* [Sam02], going through the modern and highly tweakable *Portage* [Fou99] to the (allegedly) multiplatform *Pkgsrc* [Fou97] or the very promising *XBPS* [Xtr02], several have tried to build an easy to use, community-driven, simple, with good dependency-handling, optimal, reliable, generic and portable packaging system.

In my experience on both sides of the infrastructure, as a sysadmin and developer, none of them work as one would like to. Graft, like the module system, requires manual compilation of each package, while the others aforementioned will sooner or later collide with the distribution’s package manager, causing troublesome conflicts with the system’s libraries. Keeping packages updated in the base system and/or having proper encapsulation when older versions are needed (see possible solutions below) should solve those issues.

But first, let’s explore what several HPC centers have adopted as a solution and most importantly, explore different alternatives that could substitute the *module* system [Fur96]. Let’s evaluate the good, the bad and the ugly of the module system.

The good

Widely used in different research facilities, the module system allows users to choose different versions of several software packages. The approach is simple, just type “module load program/1.0” and you can start using the version 1 of “program” as if it were installed in the system.

On the sysadmin side, it’s the same old familiar spell “tar xvfz && make && make install”, and define the module script that will set PATH, LD_LIBRARY_PATH or other variables. Consequently, the time required to wrap a software is minimal, conferring sysadmins with allegedly quick software installation. After all, efficiency in research does matter [Cuf11].

Moreover, user-coded modules can be shared easily within the same cluster by simply tweaking MODULEPATH variable. What is the catch? Each manually tweaked piece on a complex system, without properly automating its deployment, incurs in the so-called Technical debt [Wik11e]. Technical debt is about trading system robustness for quickly solving a particular issue, such as installing a package manually and fixing some related configuration file. A systematic lack of auto-

matic package management, locks sysadmins into installing each individual update manually for several packages.

The bad

Software packaging is a time consuming task that shouldn't be kept inside institutional cluster firewalls, but openly published. Indeed, a single program could have been re-packaged a number of times on each academic cluster for each university department that has HPC resources. When new versions come up for each package the sysadmin has to take care of increasing the version number (bumping it) by creating directories and additional recipes. The time investment required is hardly justifiable by today's standards and available technology.

From a technical perspective, using package systems that are not shipped with the operating system introduces an extra layer of complexity. More often than not, updates on the base distribution will break compiled programs that rely on old libraries. Stacking package managers should be considered harmful.

Ruby, Python and Perl have their own mature way to install packages for most UNIXes. Encapsulating (or stacking) package managers by, for instance, bundling a Python module inside a rpm package can potentially cause version conflicts and updates will invariably be delayed by (re-)packaging overhead. Granted, there are some concerns on uniformity, updates and security. Indeed, every UNIX operating system vendor wants control over packages, but letting individual package managers do what they do best, installing their own packages, should be the preferred option.

Another scenario where the module system would be unnecessary is cloud computing. One would have to install all the modules, and re-package the software for the virtual instances. Discarding the module system and using existing package systems, be it the Redhat Package Manager (RPM), Debian software package (deb), Python package installer (pip), Ruby package gems (gem) among others solved that by themselves. On top of that, the module system will tweak crucial system variables such as \$PATH or \$LD_LIBRARY_PATH with bad side effects for Python virtual environments or any other user-defined PATHs.

Lastly, from a human resources perspective, writing modules does not add value or expertise to your IT toolbox. On the other hand, a simple query on search engines using the keywords "job experience packaging software rpm deb" results in a flood of job offers in the software packaging area. Furthermore, being involved in open source communities via packaging can provide valuable insights on how open source projects work.

The ugly

With aging and relatively non-maintained software, bugs show up. As an example, under some circumstances, here's what occurs:

```
$ modulecmd bash purge
*** glibc detected *** modulecmd: free(): invalid next size...
```


2.1. TOOLS FOR BIOINFORMATICS DEVELOPERS

```
===== Backtrace: =====  
(...)
```

```
*** glibc detected *** modulecmd: corrupted double-linked list
```

The above error messages indicate problems with memory and data structure management from within the module system. As a result, if one runs `modulecmd` with a SUID bit [Wik11d] enabled, it can have potentially severe security implications. It is known that bad pointer management can lead to buffer overflows [Wik11b], which can be exploited maliciously. The implications of a successful compromise range from data deletion, unauthorized access to protected data or disruption of the service.

Possible future solutions on software management

Granted, the module system provides relatively easy access to older versions of software, but there are some other options that might help accomodate user's needs such as rapid novel software inclusion and version provenance:

1. Getting packages accepted in the original vendor (upstream) and/or create own package repository.
2. State the software versions you are running in your journal publications via simple reporting facilities such as “pip freeze”, Python's way to list installed modules and their currently used versions. Alternatively, one could simply resort to options shown in points below.
3. Use CDE [ED11], a lightweight application virtualization for Linux, `rbenv` [Ste11] and `virtualenv` [Gui] before and after publishing for isolation and reproducibility. Both `rbenv` and `virtualenv` create virtual Ruby or Python environments (respectively), isolated from the global HPC environment.
4. Use and store whole virtual machine images in order to better reproduce experiments. One should carefully evaluate the tradeoff between convenience and raw performance when using this solution since I/O can suffer under some circumstances.
5. If the module system is really needed, at least publish the modules somewhere for others to reuse them.
6. Use `eFfing Package Management (FPM)` [Sis11], a quickstart way to package your software into the major linux distributions (using RPM and DEB).
7. Use a configuration management system that defines (in a declarative way) the infrastructure as source code. Whole infrastructures can be blueprinted in source code and instantiated with a simple source code repository, an application data backup and compute resources. Such systems allow to automate

deployment of infrastructures in a programmatic way. Two of their most prominent implementations are Puppet [Kan03] or Chef [Ops09].

In this thesis, due to technical limitations, only Python’s `virtualenv` is used, but all the other options could be used outside traditional HPC environments.

In addition, in order to alleviate system administrator workload when installing newly appeared bioinformatics software packages, module sharing via github has been setup at the organization level (as stated in point 4 above). In this way, both users and system managers can contribute modules of common interest to a much wider audience. In our particular case, we published in-house modules at <http://github.com/scilifelab/modules.sf.net>.

2.1.4 Continuous integration system

Since some of the components detailed in chapter 3 have software tests as quality control checks, it makes sense to run all tests everytime a change is introduced in any of these systems, automatically. The purpose of a continuous integration system is to ensure that all modules continue operating correctly after any of the developers has introduced changes.

After testing Buildbot [MJ09], a Python based approach to continuous integration, a simpler alternative, Jenkins (formerly known as Hudson) [Mic09] was chosen. After some simple configuration, Jenkins monitors a remote repository and as soon as changes are detected, those are fetched and tested against the software’s testsuite, potentially notifying developers if breakages are detected.

2.2 Contributing code

There’s a general common practice in science to keep source code in private repositories, in an attempt to protect intellectual property and possible plagiarism [Mer10b] before publication. A considerable effort is made to ensure that no fraudulent data or simply copied material makes it through journals [Nat10]. However, there’s evidence showing that the rules may not be the same when it comes to scientific source code. According to source code repository logs and search engines, many bioinformatic software packages were published several months before acceptance [Bio11a].

Therefore, publishing code and contributing it back to authors has been a driving factor on the development on this thesis. In addition, an active effort has been made to teach staff on how to use this collaborative framework, as outlined in section 2.1.1. In general, the practices and techniques shown in this chapter can be assimilated and reused for other projects within the facility.

Contributing code in a public and open scenario brings many benefits that scientists are often unaware of [Bar10]. As an example, code review, commit-level commenting and unexpected collaborators around the world all help on improving

2.2. CONTRIBUTING CODE

programming skills and ultimately improve the quality of the code base, as stated in [GWCY11]:

“Making open source code publicly available is an important and often overlooked step for many projects and must be encouraged by funders, not only for the sake of community involvement in a project, but to allow others to maintain and further develop the software should a project’s funding conclude.”

Together with a proper online scientific community code of conduct [DMS⁺11] when interacting with other developers, contributing to science becomes a pleasant and continuous knowledge exchange.

Chapter 3

Pipeline integration

Given the scenario exposed in chapters 1 and 2, it is time to apply the learned insights and put them into practice. Once again, given the extensive corpus of already existing software, integration, interoperability and reuse are of utmost importance in this chapter. My focus when contributing back to the open source community has been on extending and integrating new functionality to third party software, effectively allowing to perform tasks that were not possible before.

Firstly, an assessment on how resources are managed is needed, that is, how basic computational units are handled should give us an idea of how the infrastructure works: storage, network and processing.

Secondly, a short explanation on how the Distributed Resource Management Application API (DRMAA) [TRHD07], a generalized resource manager, can enable higher levels of integration and my modifications to it are exposed.

Lastly, bcbio-nextgen, the pipeline used that will be detailed on section 3.6. My work on integrating bcbio-nextgen with our computing resources at Science for Life Laboratory, both Stockholm and HPC resources via UPPMAX, has enabled new exciting, productive and simplified ways to launch bioinformatics computing workflows.

3.1 Storage

The data generated by sequencing machines flows through a three-tier storage structure. The instruments generate raw data from the biological samples, which is pre-processed and immediately sent to the HPC facility for analysis. After the scientists have analyzed the data, it can be stored for future retrieval. There are many personal contributions from the author of this thesis to the different layers of this setup, all of them with the ideas of automation and robustness in mind:

1. Raw data sink and pre-processing machines.
2. Parallel filesystem on the HPC environment for data analysis.

3.1. STORAGE

3. Long term storage, for archival purposes.

The reason behind using a pre-processing machine is that both raw data and pre-processing steps generate immense amounts of small files in the Illumina™ platform, inside *run* folders (a *run folder* is where the sequencing machine stores the raw binary data from DNA sequencing). As seen in figure 3.1, each successful paired end run folder holds around 500,000 small files.

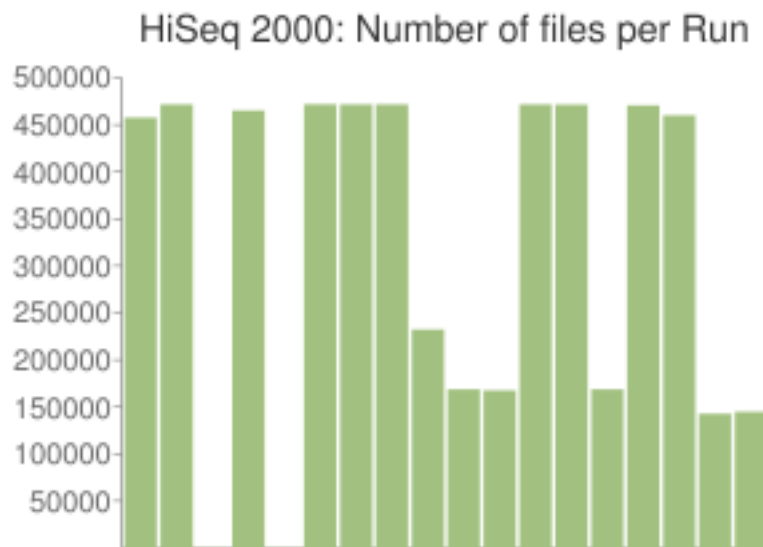


Figure 3.1. Number of files generated by run folders. The empty columns represent immediately failed runs, other runs failed halfway. If a run is successful, it invariably generates around 500,000 files.

This fact does put extra pressure on the distributed filesystem inode pre-allocations on Lustre [Wik05] systems, for instance. As an example, in our particular setup and according to official documentation, Lustre should handle up to 6 million files. In our setting, that limit was surpassed and inode exhaustion occurred, leading to filesystem outage (see last `umount` message), as illustrated in figure 3.2.

```
$ dmesg
(...)
LustreError: 11316:0:(lov_obd.c:478:lov_disconnect_obd()) Target cfs-1-OST0003_UUID disconnect error -110
LustreError: 11316:0:(ldlm_request.c:1025:ldlm_cli_cancel_req()) Got rc -108 from cancel RPC: canceling anyway
LustreError: 11316:0:(ldlm_request.c:1025:ldlm_cli_cancel_req()) Skipped 30976 previous similar messages
LustreError: 11316:0:(ldlm_request.c:1587:ldlm_cli_cancel_list()) ldlm_cli_cancel_list: -108
LustreError: 11316:0:(ldlm_request.c:1587:ldlm_cli_cancel_list()) Skipped 30976 previous similar messages
Lustre: client ffff81122eeb5400 umount complete
```

Figure 3.2. Linux kernel ring buffer backtrace on Lustre filesystem inode exhaustion. The distributed filesystem had to be unmounted to grow its inode allocation space.

In other words, in our experience, it is desirable to perform operations that reduce the number of files instead of directly sinking the raw data as it is into a distributed filesystem. Another reason to go for this strategy is to avoid single points of failure should the HPC go offline for a period of time.

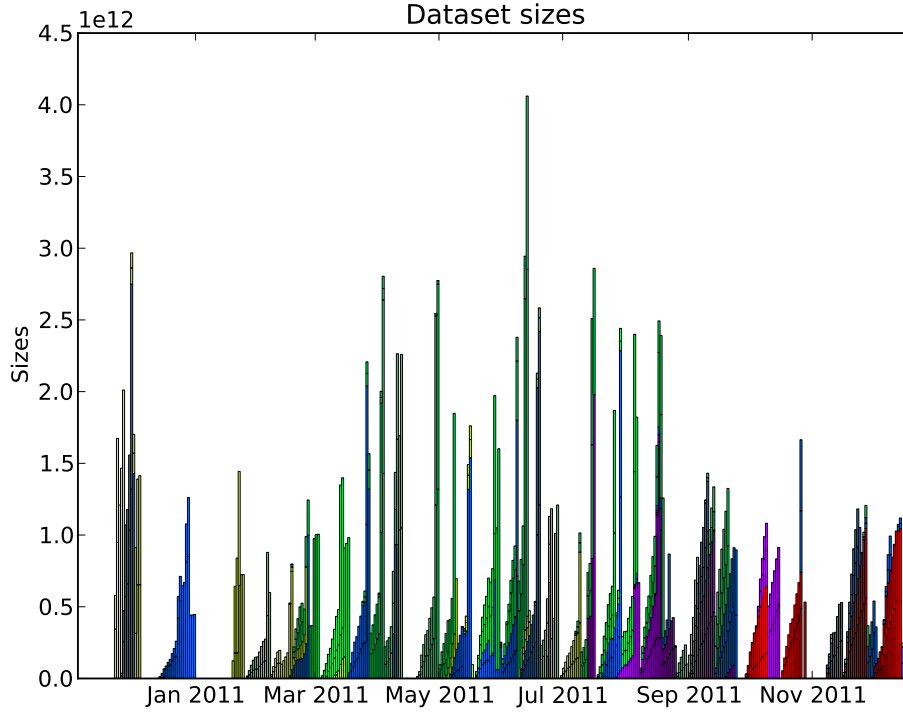


Figure 3.3. Daily disk occupancy graph on pre-processing machine, sizes in Terabytes. The different colors indicate the machine being used for a particular run. Stacked bars indicate the datasize growth of two flowcells per instrument.

In figure 3.3, a plot of the data generated by the sequencers indicates the demanding storage requirements. The Y axis shows sizes in Terabytes and the different colors represent data being generated by different instruments. As can be appreciated in the graph, by the end of August, a decision was made to decouple storage and pre-processing, dampening the requirements on storage on the computer that is directly connected to the sequencers.

When it comes to tier 2, analysis storage is a limited and expensive resource where several intermediate files are generated and speed is crucial to speedup analysis. Because of those intermediate files, format conversions and other operations, extra space and fast access is needed.

On the other hand, long term storage (tier 3) is meant to store data that has been already analyzed. A good strategy is to compress everything that is not being

3.1. STORAGE

actively used for analysis. Currently the whole run folder is archived (tar) and compressed (bzip2).

In the future, one may consider splitting datasets and enriching them with meta-data [RMH⁺10], in order to facilitate quicker access to parts of the same run.

Compression

Finding a good compression strategy is crucial to save on time and resources. As a first approach, the raw data coming directly out of the instruments was archived in a tar file and compressed with bzip2 in the pre-processing machine.

Preliminary tests on compressing datasets as they come out of the sequencer show that this approach leads to unacceptable performance. The command below uses standard unix “time” command line to measure wall clock time when compressing a 148GB run folder as it is produced from an IlluminaTM Hiseq 2000, which contains an average of 500.000 files, most of them binary basecall files (or .bcl). Later on, FastQ [CFG⁺10] files were generated and kept instead of the multiple files that compose the raw instrument output. Since FastQ files are considered more platform-independent, retrieval of data sets from the long term storage will, in theory, allow for easier access to data generated in the past.

```
$ time tar cfj 148G.tbz2 run_folder
real    697m35.334s
user    576m56.488s
sys     9m31.568s
```

It took almost **11 hours** to archive and compress 148GB with tar and bzip2 on the pre-processing machine¹. Furthermore, the resulting filesize is 101GB in size.

This relatively poor compression ratio ($101/148 = 0.67$) can be explained by the binary nature of the .bcl files, which require 1 byte per base pair².

The currently implemented solution is to use *pbzip2*, a parallelized yet bzip2 compatible version and a script we wrote and disseminated for use on our cluster environment. Using this approach the time spent on archiving and compressing a run is cut down to **2 hours**³ on average.

While tar with bzip2 performs much better when used against FastQ files, it comes with a cost. Since the file indexing metadata and payload is compressed together, no efficient random access can be performed within a tar archive. In other words, locating a small file in the compressed run has to be performed seeking the file contents linearly.

A future scenario that copes with the problem of compression versus easy access to datasets would be to use reference genomes as the CRAM toolkit [HLCB11] does.

¹SciLifeLab node (comicbookguy): Intel Xeon CPU E5630 at 2.53GHz, 8GiB RAM

²Bits 0-1 represent the base in the set {A,T,G,C} while bits 2-7 contain the quality score, according to IlluminaTM documentation

³Uppmax node: Intel Xeon CPU E5520 at 2.27GHz, 24 GiB RAM

CRAM compression ranges from 0.69 to 0.32 bits/base which is remarkable when compared with the current tar with bzip2 approach (6 bits/base). Unfortunately, at the moment of writing, CRAM 0.3 is still not recommended for production use and only a few tools support the CRAM format (samtools and Picard). In addition, CRAM requires management of reference genomes alongside, as opposed to simpler text compression achieved with general purpose algorithm.

Even if a general compression format was used, like gzip, integration at application level is still a work in progress since not all bioinformatic tools support compressed data formats. Using standard UNIX pipes to redirect the compressed stream to the program's standard input could be an option if all software supported standard input feeding. Alternatively, regular UNIX *fifo* pipes can be used instead.

Last but not least, some datacenters have adopted is compressing the data at the filesystem level using ZFS [Bon06], for instance. On the other hand, few filesystems have builtin and transparent compression features and if they do, a performance impact is to be expected unless dedicated hardware is used to offload (de)compression tasks.

3.2 Processing

In figure 3.4, we can see how the load average [And09] is lower starting on September. Unix load average is a frequently used metric to assess how loaded a system is. This number should be kept within a reasonable operational range in order to have smooth processing and avoid negative performance impacts on concurrent processes running on a computer.

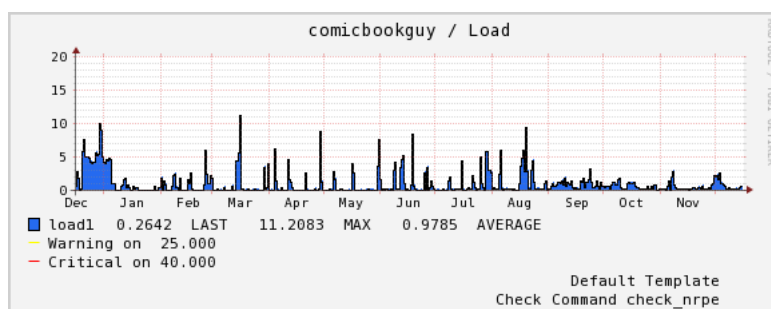


Figure 3.4. Daily load average graph on pre-processing machine.

In this particular case, since the computer is connected directly to the sequencing instruments, it is highly desirable to have a smooth operation, otherwise the sequencing run itself could get affected or unnecessary delays would be introduced in the pipeline.

The strategy we followed to lower the load average to acceptable levels was the separation of data storage and processing roles into different machines. In other words, starting on September, a decision was made to sink sequencing data on one

3.3. NETWORK

computer and do the processing on another, and that can be appreciated in figure 3.4.

3.3 Network

Empirical network tests ran by Scilifelab IT department using the *iperf* network profiling tool revealed the actual bandwidth metrics available between machines in the current infrastructure setting, as depicted in figure 3.5.

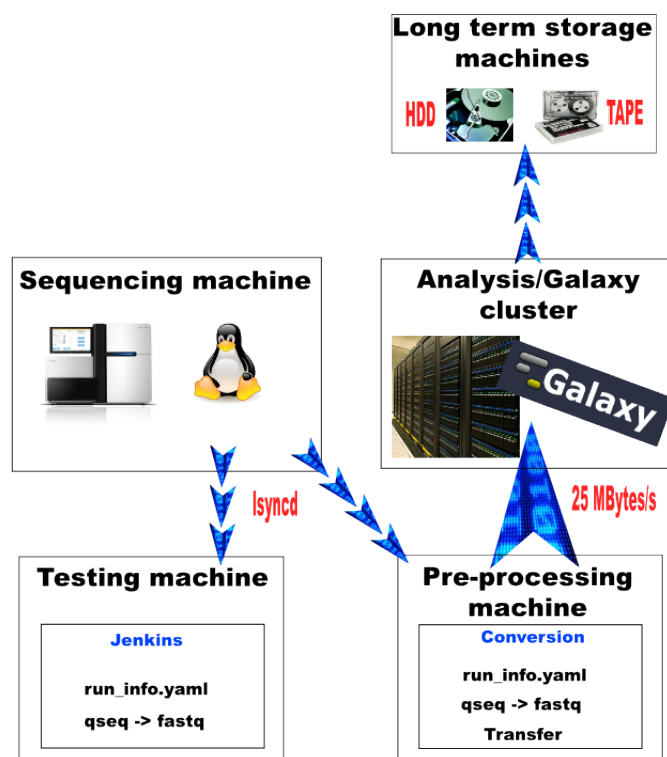


Figure 3.5. Overall architecture diagram illustrating the different storage tiers, their relations and some general inner processing tasks.

The data flows from the instrument (DNA sequencer) to a linux machine with a mounted SAMBA drive. In that machine, *lsyncd* (explained in 3.3.1) continuously synchronizes data to pre-processing and testing machines. In figure 3.6, the network usage pattern switches from August, when *rsync* synchronized to another processing dedicated machine and was later substituted by *lsyncd*. Furthermore, the substitution from hourly cron-based *rsync* transfers for the *lsyncd* daemon, allows for:

1. Better network usage: A continous stream as opposed to a batch-controlled transfer system exerts less pressure to the network.

2. Timely updates: Since new data is continuously monitored and sent immediately, less delays or idle periods are introduced in the pipeline.

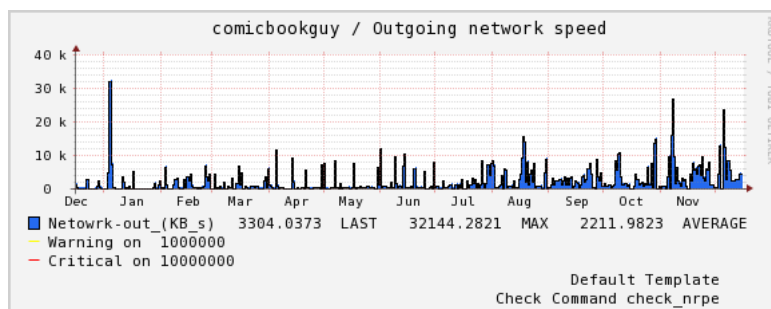


Figure 3.6. Daily outgoing network speed graph on pre-processing computer.

3.3.1 Lsyncd

The *lsyncd* daemon relies on a *kernel* based notification system to report when a file or directory has changed and act accordingly. As an example, one can imagine a user finishing a document and closing the file she was working in. In that context, *lsyncd* will detect that the user has finished writing on that file and will execute an action on that file *immediately*.

These immediate notifications on data status can be used to transfer files as soon as they are closed (or completed). That is precisely what *lsyncd* does in our genome sequencing center, once a file has been shipped by the DNA sequencer, it is sensed by the filesystem (*kernel*) and transfer immediately to other machines for further processing.

The daemon was put into production after fixing some issues⁴ that were subsequently reported upstream, to the original *lsyncd* author, following the guidelines exposed in section 2.2, so that other system developers could benefit from the fixes.

3.4 DRMAA connector

As described in section 1.3.1, computing at large scale is a rapidly changing world. Therefore, if one wants to design software that survives changes easily, different abstractions and generalization efforts are needed.

One of those abstractions is DRMAA [TRHD07]. As shown in the software stack in figure 3.7, DRMAA provides a software layer abstraction for several cluster job schedulers. It provides easy programmatic access to a generalization of batch management systems that typically runs on computational resources around the world.

⁴<https://github.com/axkibe/lsyncd/issues/93>

3.4. DRMAA CONNECTOR

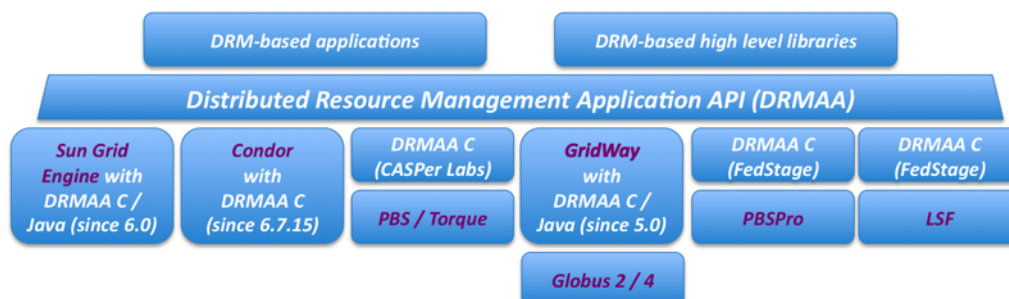


Figure 3.7. DRMAA stack figure from drmaa.org website. SLURM batch system would be at the same level of the other job managers (Sun Grid Engine, Condor, etc. . .). DRMAA C/Java refers to the different language implementations available at this time for the interface with the respective job managers.

In particular, the batch management system that DRMAA talks to during this thesis is SLURM, so efforts were put on the integration between those two components whose controlling library is called “slurm-drmaa”.

My task with the slurm-drmaa connector consisted on working with the low level library that interacts more closely with the SLURM queue system and spot deficiencies that prevented to launch jobs through the stack and ultimately in the cluster. In particular some crucial native SLURM attributes used in some facilities were unsupported, as well as shell environment migration between nodes. As proposed in chapter 2.2 the additions were subsequently reported upstream to the original authors and included for general public use.

This apparently minor contribution on two open source libraries (slurm-drmaa and python-drmaa) opens new doors to other centers that use SLURM as a queue job manager to run a variety of third party HPC applications in a general way. As an example, prior to this effort, Galaxy was unable to run on SLURM systems, albeit Galaxy supported several other job managers through this abstraction.

As a proof of generality for the python-drmaa approach, the following script screens datasets in search for organism contamination using a popular Perl script, called “fastq_screen”, developed by Babraham Institute⁵. It does so by setting up the job attributes, launching the job and destroying the session for each dataset. This simple script can be generalized as an API, in order to avoid writing SLURM-specific “sbatch” shell scripts, as shown in listing 3.1.

```
#!/usr/bin/env python
import drmaa
import os
import sys

# A set of biological samples in FastQ file format
samples = glob.glob(sys.argv[1]+"*.fastq")
```

⁵http://www.bioinformatics.bbsrc.ac.uk/projects/fastq_screen/

```

for sample in samples:
# DRMAA initializations
    s = drmaa.Session()
    s.initialize()

# Actual command and arguments
# to be launched to the cluster
    jt = s.createJobTemplate()
    jt.remoteCommand = 'fastq_screen'
    jt.args = ['--illumina', '--multilib', '--subset',
               '2000000', sample1, '--paired',
               sample2, '--outdir', '.']

# Modifications contributed upstream,
# allowing to set different fine grained
# aspects on particular cluster setups
    jt.nativeSpecification = "-A_project -p_node -t_00:30:00"

    jobid = s.runJob(jt)
    print 'Your_job_has_been_submitted_with_id_' + jobid

    s.deleteJobTemplate(jt)
    s.exit()

```

Listing 3.1. Example python code using the python-drmaa connector. It submits a genomic contamination screening program to the cluster job manager, in our case, SLURM.

Further attributes need to be added to the respective implementations, but as it is now, the provided patches capture the minimum DRMAA 1.0 requirements to submit jobs to the cluster⁶.

3.5 Blue Collar Bioinformatics pipeline

The bcbio-nextgen nextgen pipeline developed by Brad Chapman was chosen following guidelines proposed on 1.4, but more specifically:

1. For its effort to integrate several already present bioinformatic tools together, where Python is the glue that binds them all [vR98].
2. Being open source project hosted on Github, welcomes contributions from different sources.

⁶Minor details such as job name and other native specification attributes need to be refined in the future for better sysadmin accounting.

3.6. HOW DOES BCBIO-NEXTGEN WORK ?

3. Code experiments should be generalized and published for reuse and automation.

All the other points referred before on 1.4 are present in this pipeline as well, with advantages and shortcomings for each of them.

3.6 How does bcbio-nextgen work ?

The pipeline consists of about 4000 lines of Python code that bind together the typical steps followed by informaticians when performing exome sequencing analysis [PVP⁺12, Bio11c]. The pipeline can also perform other types of analysis (RNA-Seq, for instance) if parameters are provided adequately. Exome sequencing, that is, selectively sequencing coding regions of a genome, will be described since is a common practice within our facility.

A brief explanation on code structure is published as part of the official documentation parts of it were contributed by the author of this thesis ⁷.

3.7 Upstream pre-processing

A script called `illumina_finished_msg.py` is tightly coupled with the sequencer, as it detects the hints or termination signals when a sequencing run has finished. The “run termination signals” that the instrument provides, be it plaintext or xml formatted files, vary among instrument updates. As a result, this script has evolved to handle the cases with most of the IlluminaTM GAII and different HiSeq2000 revisions.

A scheduled process (cron) calls the aforementioned script, looking for a termination of a sequencing run. When a run is finished, the instrument (sequencer), leaves specific files in the filesystem. Those hints will be used to kick-off further actions when present. Other present and future platforms (SOLiD, 454, PacBio, etc...) could potentially use this script as a template to detect and pre-process data.

Moreover, a state file, referred by `msg_db` in `transfer_info.yaml` keeps track of the runs that have been processed in the past, ignoring them if already processed. On the other hand, new runs not present in the state file, undergo transformations from binary `.bcl` format to the more standard FastQ format [CFG⁺10].

After the FastQ files for each lane have been generated, two messages are sent to the RabbitMQ queuing system: `store` and `analyze`. Those messages will be consumed by another running script called `analyze_finished_sqn.py`.

Again, this script is platform-specific and should be used as a template for other instruments. What it does is scan through the `qseq` files generated on the previous step and generate a single FastQ file per lane⁸.

⁷Available at: <https://github.com/SciLifeLab/bcbb/blob/master/nextgen/README.md>

⁸A “lane” is a subunit of an IlluminaTM run.

3.8 Downstream analysis

The software used in the so-called “downstream analysis”, happening at the cluster, is composed out of several commonly used bioinformatic tools used as format converters (picard), aligners (bwa), SNP callers (GATK) and SNP effect predictors (snpEff). All the programs and parameters can be configured with varying amounts of effort. A general overview⁹ of the steps involved in a typical exome sequencing project, such as, for example spotting the genetic cause of a mendelian disorder [NBL⁺10, PVP⁺12]. A simplified view of the steps performed by this type of analysis, using our pipeline, is outlined below:

1. Align FastQ file to reference with *bwa* → file.sam
2. Sort and convert to bam using *picard*:
 - a) *picard* FastqToSam → file.bam
 - b) *picard* MergeBamAlignment → file.bam
 - c) *picard* SortSam → file-sort.bam
3. If read pairs, merge pairs file-sort_1.bam and file-sort_2.bam to file-sort.bam with *picard*’s MergeSamFiles class.
4. Mark duplicates with *picard* MarkDuplicates → file-sort-dup.bam
5. Recalibrate file:
 - a) *GATK* -T CountCovariates → file-sort-dup.recal
 - b) *GATK* -T TableRecalibration → file-sort-dup-gatkrecal.bam
 - c) Recalibration quality report
6. Realign indels, run genotyper and filter variants:
 - a) *GATK* -T IndelRealigner → file-sort-dup-gatkrecal-realign.bam
 - b) *GATK* -T UnifiedGenotyper → file-sort-dup-gatkrecal-realign-snp.vcf
 - c) *GATK* -T VariantFiltration → file-sort-dup-gatkrecal-realign-snp-filter.vcf
7. Evaluate genotypes
 - a) *GATK* -T VariantEval → file-sort-dup-gatkrecal-realign-snp-filter.eval
8. Calculate variation effects
 - a) `java -jar snpEff.jar` → file-sort-dup-gatkrecal-realign-snp-filter-effects.tsv

3.8. DOWNSTREAM ANALYSIS

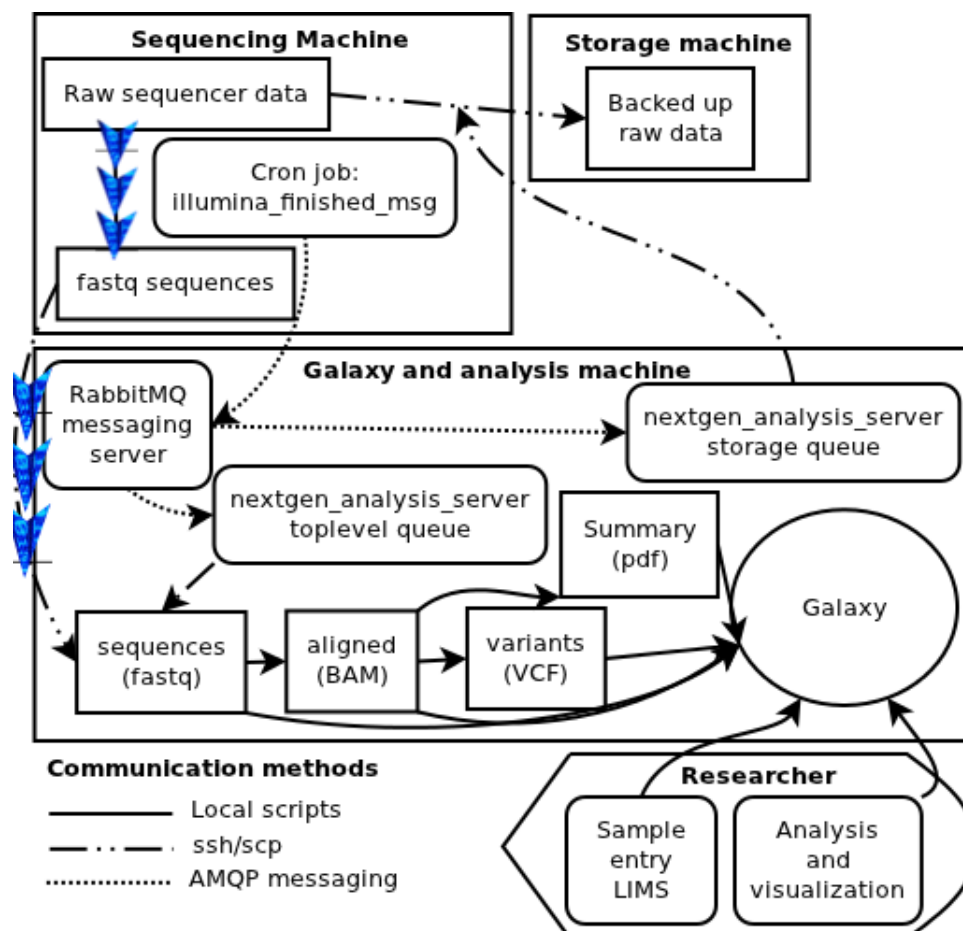


Figure 3.8. bcbio-nextgen general systems architecture

A general overview is shown in figures 3.5 and 3.8, but for extra insight it is interesting to elucidate what the different scripts do and how they act together.

Another script, `analyze_finished_sqn.py`, runs continuously in the background, consuming messages from the RabbitMQ queue¹⁰. Once a message is consumed, it decides where and how the run should be stored and analyzed.

For additional reliability, this script is run under the supervision of *supervisord* daemon, which monitors the state of the process and respawns it if it's accidentally killed or crashes unexpectedly.

When the FastQ files have been transferred, the `automated_initial_analysis.py` script prepares the lanes and kicks off the demultiplexing program, defined in the `post_process` configuration file. Depending on the directives supplied to the run configuration (`run_info.yaml`), the automated initial analysis can perform different

⁹Thanks to Per Unneberg for synthetic step-wise summary.

¹⁰A queue implementation based on the Advanced Messaging Queuing Protocol (AMQP).

types of analysis.

3.8.1 Samplesheet

A basic piece of metadata that details how a sequencing run is defined is the samplesheet. The samplesheet is a CSV file that specifies which barcodes belong to specific samples, project metadata among other fields. As an early approach to pipeline-to-LIMS integration, this file was chosen as an initial Rosetta stone (central meta-information provider) for metadata processing. Different extra functionality, test coverage and documentation has been contributed as part of this thesis regarding this component.

The samplesheet is converted to a more flexible YAML format that, in addition to the data already present on the previous CSV file, adds information on which type of analysis the specific samples will undergo. The file generated after this conversion is named `run_info.yaml` and can be understood as a serialized representation of the sequencing run.

3.9 Galaxy

The official Galaxy resource pages that outline how to deploy Galaxy in a production cluster environment already make good points on how to properly install Galaxy¹¹. Those recommendations involve:

1. Using a clean environment, and a dedicated user account.
2. Scaling up the local SQLite database to a dedicated server instance (such as PostgreSQL), allows for concurrent connections to the metadata, speeding up the response of the whole system.
3. Enable mechanisms to send data off-browser (FTP, GridFTP or others).
4. Different performance tuning aimed at providing a better user experience.

This section will deal with the perceived security risks in our particular environment alongside different options available to run Galaxy and their configurations.

3.9.1 Galaxy deployment on HPC

From a security standpoint, there are three main concerns that should be addressed [Gui11a] when running Galaxy in multi-user HPC systems:

1. Target users: State which users are allowed to use the service. In other words, assess the risk of letting users use a particular service.

¹¹<http://usegalaxy.org/production>

3.9. GALAXY

2. Accountability: Track and record what actions users have performed in the system (logs).
3. Security issues: Bugs in the software that can cause misuse of the HPC resources.

Point 1 boils down to the terms of service (TOS) of the HPC resource itself. In other words, using Galaxy should be regarded as running any other tool in a cluster. Therefore, as long as the user has agreed with the TOS and accepted full responsibility (and liability), the same rules apply to any other users in a cluster willing to run any type of software.

Secondly, when running Galaxy, there is the possibility to enable *logging* for user actions. This user monitoring mechanism can be backed by user authentication used by the underlying method used by the HPC facility. As a result, Galaxy can attain accountability levels superior to those standalone tools which are not subject to process accounting through *psacct*¹².

Lastly, verified security issues such as improper input validation and the resulting exploitations of those flaws should be reported upstream as soon as discovered. As one would expect, any malicious usage of those bugs are subject to the same rules exposed in point 1.

Having discussed the delimitations and common concerns on running new software on HPC systems, we can proceed and describe possible deployments.

3.9.2 Tunneled Galaxy access

In its simple use case, Galaxy can be spawned as a so-called “developer instance”. This option was designed to allow a software developer to deploy a galaxy instance on a local computer minimizing configuration hassle. This approach, coupled with a standard SSH tunneling¹³, comes in really handy when testing and debugging new features in galaxy [Gui11a]. Figure 3.9 depicts how a Galaxy instance is presented to the user via a web browser while the actual service and data is running on a remote cluster. All Galaxy’s user interaction via a standard browser that supports HTTP, is encapsulated inside a SSH tunnel which connects to a HPC resource at the remote end.

However, this is not an option when more than one user is supposed to work concurrently on this system with optimal performance and responsiveness.

Pros:

1. Relatively easy to setup, for a knowledgeable user.

Cons:

¹²A well known UNIX process accounting tool

¹³Performed by the UNIX command: `ssh -L 8080:localhost:6666 username@HPCfacility`

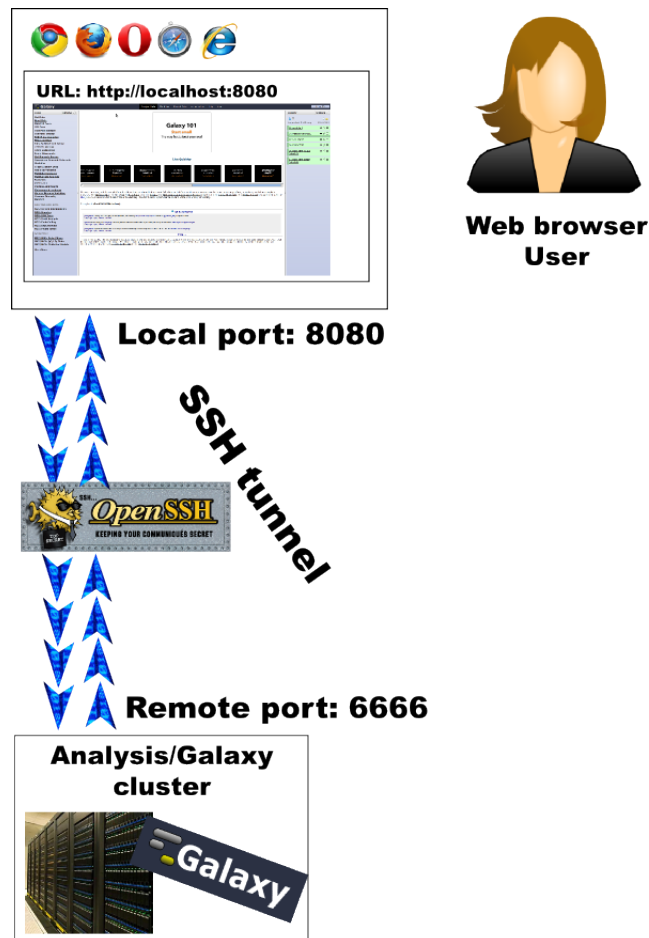


Figure 3.9. Established secure SSH tunnel between client (web browser on user's machine) and server (HPC resource). The user connects to localhost, but the Galaxy instance runs at the other end, in a cluster.

1. Running a single-thread, Python web server (paster), noticeable impact on performance.
2. Running a serverless database that does not handle concurrency optimally.
3. Runs a single galaxy process, inefficient due to the Python's global interpreter lock (GIL), the way Python handles threading at the moment.
4. Makes sharing of datasets and libraries unnecessarily difficult, creating data islands.
5. User accounting delegated to the user. Meaning, it can be tampered at any time, exposing a personal resource (a UNIX account in this case), to third parties.

3.9. GALAXY

In the latest galaxy community conference (GCC2011) a Galaxy production setup was presented, the main Galaxy public instance (<http://usegalaxy.org>). A graphical example on how the infrastructure looks like is shown in figure 3.10.

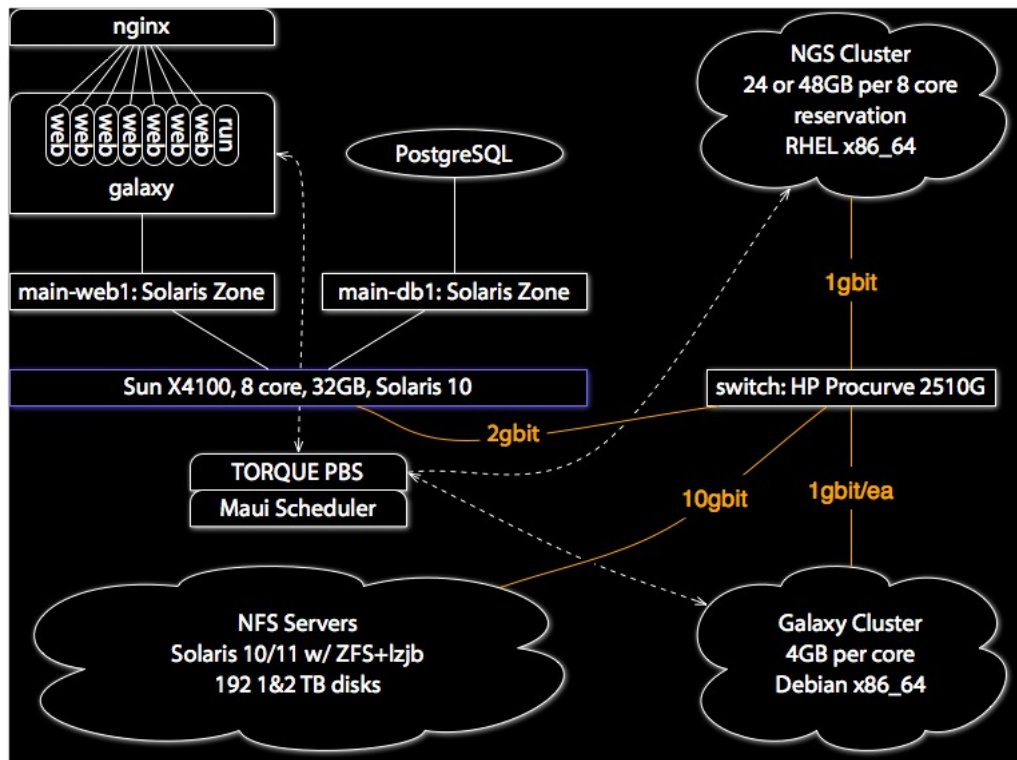


Figure 3.10. Optimal production setup (<http://usegalaxy.org/production>), as it is for the Galaxy “main” instance at Penn State University.

The approach followed here is to have a dedicated “galaxy” user account for the whole galaxy installation. On the other hand, this approach requires a component to have superuser privileges (SUID) in order to switch between the multiple cluster users.

This restriction could potentially open a huge security hole on the system, since uncontrolled execution flows can lead to full control of the whole cluster so extra care must be taken when considering which component should be used to switch users via `setuid`. Putting it simply, the decision on which piece of code switches between users is purely based on trust, so both users and system administrators should seek common ground when deciding code privileges.

At the time of writing, there are two ways to run galaxy while integrating with an existing authentication on clusters with minimally intrusive additions: Apache-based modules and DBMAA runner.

3.9.3 Apache

Two plugins can be added to apache to allow authentication for UNIX accounts: *mod_authz_external* and *pwauth*. Those will use the accounts already present on the system to both authenticate against galaxy’s web interface and run tool commands.

Again, this approach requires an executable to change between users, to have the SUID bit enabled. Furthermore a long discussion on how to balance the risks and to mitigate those is already provided. It falls to the hands of the sysadmins to handle this risk scenario in the best manner, taking into account the specifics of the cluster system one is running.

Moving the SUID bit requirement down to just the program that actually runs the jobs on the cluster allows us to better isolate the security risk that it entails.

This is the preferred solution that is being developed by the galaxy community, including Illumina™.

Pros:

1. Sharing a central galaxy instance eases administration and updates.
2. It does not use extra ports on the server for each user.
3. Increased performance and responsiveness.

Cons:

1. More efforts should be put on the infrastructure deployment by system administrators, namely: Galaxy and database, performance tweaks, load balancing and other robustness improvements.

Despite many inconveniences shown with the tunneled approach over a central instance, the current supported alternative at our particular HPC facility is the tunneled Galaxy access 3.9.2.

Nevertheless, the tunneled Galaxy option at least allows for development and testing of new tools within the Galaxy framework. As an example, a commandline proteomics tool has been used as a proof of concept, as explained in section 3.10.

3.10 Empirical tests

On one hand, a particular proteomics tool, Percolator [KCW⁺07], developed by Lukas Käll group was preliminarily implemented, tested and committed as a Galaxy tool, demonstrating the integration of the underlying infrastructural components described in previous sections.

On the other hand, other beta testers of the system claimed complex setting up of private user Galaxy instances. That fact eroded trust on the tools and lead researchers back to traditional ad hoc script writing. But even if custom scripts are developed, one can add those into Galaxy, or in other words:

3.10. EMPIRICAL TESTS

“If it can be run on the command line, it can be wrapped inside Galaxy”

Truth being told, wrapping a tool inside Galaxy requires some time investment that not all researchers are willing to pay for. On the other hand, having user friendly interface to a command line tool **does** require development efforts anyhow, so why not spend it in a community supported effort such as Galaxy?

Luckily, the same command line principle applies to the bcbio-nextgen pipeline, allowing automated best practice workflows. Therefore, bcbio-nextgen complements Galaxy by providing a set of predefined analysis while one can tweak parameters and re-run analysis, experimenting within Galaxy. My contributions to the base code and its integration with SciLifeLab and UPPMAX infrastructure has enabled another level of automation that was much needed in order to free up time from our core staff and researchers.

Chapter 4

Final remarks

4.1 Conclusions

The original question in this thesis work was if it is possible to automate data processing in a bioinformatics core facility. The answer is that a high throughput sequencing pipeline has been successfully extended while running in production on different sequencing centers located in different countries. Our facility, Science For Life Laboratory, has benefited in many directions from this thesis.

My personal contributions to the community and to our particular facility have been diverse but always with an integrative focus. The bcbio-nextgen pipeline has been augmented to meet our needs by defining the samplesheet as an information repository when lacking a LIMS. Several other features have been added to the pipeline and tests have been written for it.

On a infrastructural level, the DRMAA connector, with both modifications to C and Python layers, has enabled a new way to interact with SLURM based clusters, such as ours. Obviously, the improvements on the software and documentation have been communicated back to the maintainers of the different components, improving the overall state of our discipline.

Towards the high level (user) side, a proof of concept to run Galaxy on our HPC environment has been demonstrated and tested by some users. In parallel with all of the above, a substantial dissemination effort has been made to communicate the advantages of my work to our core and external groups, as seen in the appendix, with an academic poster being presented at the SeqAhead.eu conference.

Despite several time consuming manual steps being successfully automated, there's still room for improvements and new features. Nevertheless, new software additions and stable data processing are two ends that do not always meet easily. As a result, solid software development methods such as automated software testing and advanced source code sharing workflows had to be introduced in the programming culture of a bioinformatics core team.

Furthermore, these same methods allowed the staff to contribute back to bioinformatics science field, by extending already present functionality in different soft-

4.2. FUTURE WORK

ware packages. In addition, contributions on different software components allowed Galaxy, a bioinformatics workflow system, to be run in a HPC environment running SLURM job scheduling system, being actively used by other facilities around the world.

4.2 Future work

Despite successfully running Galaxy framework in a HPC environment as a proof of concept, more work is needed to disseminate Galaxy and bcbio-nextgen acquired know-how via workshops and general training, in order to get more users involved.

Future technical lines of work include successfully integrating iRODS for standardized data provenance and metadata management. For example, a future iteration of the samplesheet would be to generalize the metadata fields from this file and include them on iRODS [RMH⁺10], expose them as a RESTful API or migrate them to a LIMS or better integrated systems [RSBM⁺10]. Many more features could be incorporated in bcbio-nextgen, such as the powerful filtering tool FACS [SKA⁺10], being extended at the moment of this writing to support extremely fast contamination screening by using bloom filters.

Integrating protocols such as GridFTP, would allow increased transfer rates and better interoperability between HPC environments. Furthermore, more research and optimization is needed when transferring big datasets over networks and to make biomedical HPC and Cloud computing access totally seamless. In that line, the author is involved in two EU projects (eCloudMan and SeqAhead.eu) that aim to develop those systems in the near future.

4.3 Acknowledgements

Så här är det äntligen, färdigt. Det verkar så länge sedan jag bodde i Barcelona och undrade vad jag skulle göra i min framtid efter att ha godkänts av KTHs mastersprogram i beräkningsbiologi.

Tusen tack till alla, speciellt till **Lars Arvestad**, för hans vägledning, råd och tålamod. Till **Fredrik Lennmark** och **Eric Björkvall** för supersnabba IT-tjänster och alltid vänlig och mycket professionell feedback. Min rekryterare **Magnus Bjursell**, tack för det förtroende du hade för mig när jag började på SciLifeLab.

Också under mina tidiga steg, **Ellen Sherwood**, tack för din vägledning genom denna märkliga bioteknikvärld.

Mina chefer **Per Kraulis** och **Thomas Svensson**, för en handledningsupplevelse baserad på frihet och förtroende.

Arbetskamrater **Per Unneberg** och **Mikael Huss**, otroligt skickliga individer.

Pontus Larsson och **Valentine Svensson**, två underbara hackare med ovärderliga insikter och färdigheter. **Luminița Moruz** och **Paul Igor Costea**, mina rumänska roliga och otroligt hjälpsamma vänner.

Från Uppsala, **Martin Dahlö**, **Samuel Lampa**, **Ola Spjuth** och all UPPMAX personal för stöd till tekniska och implementationsaspekter för koden som körs på UPPMAX resurser. För att de snabbt svarar på olika drmaa tvivel, **Mariusz Mamonski** och **Enrico Sirola**, folk som jag inte känner personligen, men är inte internet en underbar uppfinning?

Alla som hade tillräckligt med tålamod för att korrigera mina nybörjare:s svenska :)

Min oväntade distansmentor **Brad Chapman**, det skulle inte ha varit en så spännande upplevelse utan sådan begåvad person.

Jag vill avsluta med **min familj**, alltid i mitt hjärta, oavsett hur många kilometer som skiljer oss.

4.4 Appendix

4.4. APPENDIX

SciLifeLab



¹ Science for Life Laboratory, Department of Biochemistry and Biophysics, Stockholm University, Stockholm, Sweden. E-mail: roman@scilifelab.se

We forked and extended an existing open source next-gen pipeline and adapted it to our core facility needs. It allows our researchers to get an unattended automatic preliminary report for all samples, including quality, alignments and genotyping. It is highly extensible, runs in both HPC and Cloud environments and makes all processed data easily available in the Galaxy web environment.

bcbio-nextgen is a Python framework for next generation sequencing analysis. The fully automated pipeline interacts with the sequencing machine, runs sequences through configurable processing pipelines, and uploads the data into Galaxy for visualization and additional processing.

The CloudBioLinux and CloudMan projects utilize this pipeline for distributed analysis on Amazon cloud infrastructure. The Galaxy web-based analysis tool can be optionally integrated with the analysis scripts. Tracking of samples occurs via a web based LIMS system, and processed results are uploaded into Galaxy Data Libraries for researcher access and additional analysis.

The diagram illustrates the Galaxy architecture, showing the flow of data and the roles of various components. The process starts with a **Sequencing Machine** generating **Raw sequence data**, which is then stored in a **Storage machine** as **Backed up raw data**. The **Storage machine** provides data to the **Galaxy and analysis machine**, which includes a **RawBAM messaging server** and a **nextgen_analysis_server storage queue**. The **Galaxy and analysis machine** processes data into **fastq sequences**, **aligned (BAM)**, and **variants (VCF)**. The **Galaxy** web interface, represented by a blue circle, is used by the **Researcher** to upload data and view results. The **Researcher** can also use **Sample entry LIMS** and **Analysis and visualization** tools. The **Galaxy** web interface is connected to the **Galaxy and analysis machine** and the **Researcher**.

```

graph TD
    SM[Sequencing Machine] -- "Raw sequence data" --> STM[Storage machine]
    STM -- "Backed up raw data" --> STM
    STM -- "RawBAM messaging server" --> GAMS[Galaxy and analysis machine]
    GAMS -- "nextgen_analysis_server storage queue" --> GAMS
    GAMS -- "fastq sequences" --> GAMS
    GAMS -- "aligned BAM" --> GAMS
    GAMS -- "variants VCF" --> GAMS
    GAMS -- "Summary pdf" --> GAMS
    GAMS -- "Galaxy" --> GAMS
    GAMS -- "Galaxy" --> R[Researcher]
    R -- "Sample entry LIMS" --> R
    R -- "Analysis and visualization" --> R
    R -- "Galaxy" --> GAMS
  
```

Sequencing Machine

- Raw sequence data

Storage machine

- Backed up raw data

Galaxy and analysis machine


- RawBAM messaging server
- nextgen_analysis_server storage queue
- fastq sequences
- aligned BAM
- variants VCF
- Summary pdf
- Galaxy

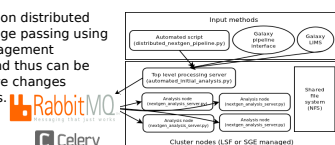
Researcher

- Sample entry LIMS
- Analysis and visualization

Communication methods

- Local scripts
- Galaxy
- Galaxy messaging

bcbio-nextgen relies on distributed asynchronous message passing using RabbitMQ. Task management is then abstracted and thus can be generalized to survive changes and temporal failures.  RabbitMQ



- The pipeline works on a wide variety of software infrastructures, as demonstrated by deployment on three heterogeneous production environments that process data coming from Illumina GAI and Hiseq 2000 sequencing instruments:

- Amazon elastic cloud computing has also been used successfully to process and analyze data in combination with CloudMan and Galaxy. This enables deployment and scaling of the system by users without existing cluster resources.

- Fully open source software collaboration via GitHub source code sharing platform allowed our respective core facilities to quickly respond to new research requirements, rapidly adapting to platform updates.

Library preparation

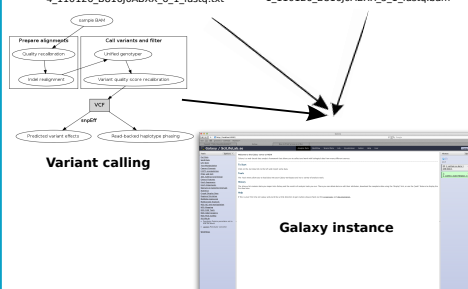
- DNA
- Library
- Sequencing

Data analysis

- Data
- Analysis
- Variant calling
- UCSC visualization

```
4_110126_B816J0ABXX_1_1_fastq.txt
4_110126_B816J0ABXX_5_2_fastq.txt
4_110126_B816J0ABXX_1_2_fastq.txt
4_110126_B816J0ABXX_6_1_fastq.txt
```

```
1_110126_B816J0ABXX_5.sam
1_110126_B816J0ABXX_5.bam
1_110126_B816J0ABXX_5-sort.bam
1_110126_B816J0ABXX_5_1.fastq.bam
```



SciLifeLab is a collaboration between four Swedish universities in Stockholm and Uppsala: Stockholm University, the Karolinska Institutet, the Royal Institute of Technology (KTH) and Uppsala University. The centre combines advanced technical know-how and state-of-the-art equipment with a broad knowledge in translational medicine and molecular bioscience.

bcblo-nextgen can be forked from <http://github.com/chapmanb/bcbb/tree/master/nextgen>. For more information please contact Roman Valls Guimera (roman@scilifelab.se) and/or Brad Chapman (bchapman@hsph.harvard.edu).

[1] <http://bcbio.wordpress.com/>
[2] <http://usegalaxy.org>

This project was supported by SciLifeLab genomics platform staff and invaluable support from Brad Chapman.

Bibliography

- [AAG10] Mohamed Abouelhoda, Shady Alaa, and Moustafa Ghanem. Meta-workflows. In *Proceedings of the 1st International Workshop on Workflow Approaches to New Data-centric Science - Wands '10*, pages 1–8, New York, New York, USA, 2010. ACM Press.
- [ABC⁺10] Enis Afgan, Dannon Baker, Nate Coraor, Brad Chapman, Anton Nekrutenko, and James Taylor. Galaxy CloudMan: delivering cloud compute clusters. *BMC Bioinformatics*, 11(Suppl 12):S4, 2010.
- [ABC⁺11] Enis Afgan, Dannon Baker, Nate Coraor, Hiroki Goto, Ian M Paul, Kateryna D Makova, Anton Nekrutenko, and James Taylor. Harnessing cloud computing with Galaxy Cloud. *Nature Biotechnology*, 29(11):972–974, November 2011.
- [ABT⁺11] Enis Afgan, Dannon Baker, the Galaxy Team, Anton Nekrutenko, and James Taylor. A reference model for deploying applications in virtualized environments. *Concurrency and Computation: Practice and Experience*, August 2011.
- [And09] Andre. Understanding Linux CPU Load - when should you be worried?: <http://blog.scoutapp.com/articles/2009/07/31/understanding-load-averages>, 2009.
- [Bar10] Nick Barnes. Publish your computer code: it is good enough. *Nature*, 467(7317):753, October 2010.
- [BCH⁺10] David B Burdick, Chris C Cavnor, Jeremy Handcock, Sarah Killcoyne, Jake Lin, Bruz Marzolf, Stephen A Ramsey, Hector Rovira, Ryan Bressler, Ilya Shmulevich, and John Boyle. SEQADAPT: an adaptable system for the tracking, storage and analysis of high throughput sequencing experiments. *BMC Bioinformatics*, 11(1):377, July 2010.
- [BDF⁺03] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. *Xen and the art of virtualization*. ACM Press, New York, New York, USA, 2003.
- [Bio11a] Biostar.stackexchange.com. Does pre-publishing open source algorithms hurt chances of getting published?, 2011.
- [Bio11b] Biostar.stackexchange.com. Is there a LIMS that doesn't suck?, 2011.
- [Bio11c] Biostar.stackexchange.com. What is the best pipeline for human whole exome sequencing?, 2011.
- [Bon06] Jeff Bonwick. ZFS, the zettabyte filesystem. In *USENIX*, 2006.

BIBLIOGRAPHY

- [BSB04] Catherine A Ball, Gavin Sherlock, and Alvis Brazma. Funding high-throughput data sharing. *Nature Biotechnology*, 22(9):1179–83, September 2004.
- [BYV⁺09] Rajkumar Buyya, Chee Shin Yeo, Srikumar Venugopal, James Broberg, and Ivona Brandic. Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation Computer Systems*, 25(6):599–616, June 2009.
- [Cal11] Ewen Callaway. Genome giant offers data service. *Nature*, 475(7357):435–7, July 2011.
- [Cat10] Daniele Catteddu. Cloud Computing: Benefits, Risks and Recommendations for Information Security. In Carlos Serrão, Vicente Aguilera Díaz, and Fabio Cerullo, editors, *Web Application Security*, volume 72 of *Communications in Computer and Information Science*, page 17. Springer Berlin Heidelberg, 2010.
- [CCQ⁺11] Gen-Tao Chiang, Peter Clapham, Guoying Qi, Kevin Sale, and Guy Coates. Implementing a genomic data management system using iRODS in Wellcome Trust Sanger Institute. *BMC Bioinformatics*, 12(1):361, September 2011.
- [CFG⁺10] Peter J A Cock, Christopher J Fields, Naohisa Goto, Michael L Heuer, and Peter M Rice. The Sanger FASTQ file format for sequences with quality scores, and the Solexa/Illumina FASTQ variants. *Nucleic acids research*, 38(6):1767–71, April 2010.
- [Cha11a] Brad Chapman. Freely available, community maintained software images and data repositories for biological analysis: <http://cloudbiolinux.org>, 2011.
- [Cha11b] Brad Chapman. Next generation sequencing information management and analysis system for Galaxy, 2011.
- [CM11] Marcin Cieslik and Cameron Mura. A lightweight, flow-based toolkit for parallel and distributed bioinformatics pipelines. *BMC Bioinformatics*, 12(1):61, February 2011.
- [Cuf11] James Cuff. Velocity in research computing *really* matters: <http://blog.jcuff.net/2011/04/velocity-in-research-computing-really.html>, 2011.
- [DB09] Joel T Dudley and Atul J Butte. A quick guide for developing effective bioinformatics programming skills. *PLoS Computational Biology*, 5(12):e1000589, December 2009.
- [Dee09] E. Deelman. Grids and Clouds: Making Workflow Applications Work in Heterogeneous Distributed Environments. *International Journal of High Performance Computing Applications*, 24(3):284–298, December 2009.
- [DG08] Jeffrey Dean and Sanjay Ghemawat. MapReduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107, January 2008.
- [DMS⁺11] Giovanni M. Dall’Olio, Jacopo Marino, Michael Schubert, Kevin L. Keys, Melanie I. Stefan, Colin S. Gillespie, Pierre Poulain, Khader Shameer, Robert Sugar, Brandon M. Invergo, Lars J. Jensen, Jaume Bertranpetit, and Hafid Laayouni. Ten Simple Rules for Getting Help from Online Scientific Communities. *PLoS Computational Biology*, 7(9):e1002202, September 2011.

BIBLIOGRAPHY

- [Doc08] Cory Doctorow. Big data: Welcome to the petacentre. *Nature*, 455(7209):16–21, September 2008.
- [Dri11] Vincent Driessen. Nvie’s GitFlow: <https://github.com/nvie/gitflow>, 2011.
- [ED11] Philip J. Guo Engler and Dawson. CDE: Using System Call Interposition to Automatically Create Portable Software Packages. In *USENIX ATC*, page 15, 2011.
- [EHT10] Erik Elmroth, Francisco Hernández, and Johan Tordsson. Three fundamental dimensions of scientific workflow interoperability: Model of computation, language, and execution environment. *Future Generation Computer Systems*, 26(2):245–256, February 2010.
- [Fou97] NetBSD Foundation. Pkgsrc - The NetBSD Packages Collection: <http://www.netbsd.org/docs/software/packages.html>, 1997.
- [Fou99] Gentoo Foundation. Portage, the software distribution system for Gentoo GNU/Linux: <http://www.gentoo.org/proj/en/devrel/handbook/handbook.xml>, 1999.
- [Fou11] Apache Foundation. CouchDB, 2011.
- [FPG⁺11] Vincent A. Fusaro, Prasad Patil, Erik Gafni, Dennis P. Wall, and Peter J. Tonellato. Biomedical Cloud Computing With Amazon Web Services. *PLoS Computational Biology*, 7(8):e1002147, August 2011.
- [FSC⁺09] Dawn Field, Susanna-Assunta Sansone, Amanda Collis, Tim Booth, Peter Dukes, Susan K Gregurick, Karen Kennedy, Patrik Kolar, Eugene Kolker, Mary Maxon, Siân Millard, Alexis-Michel Mugabushaka, Nicola Perrin, Jacques E Remacle, Karin Remington, Philippe Rocca-Serra, Chris F Taylor, Mark Thorley, Bela Tiwari, and John Wilbanks. Megascience. ‘Omics data sharing. *Science (New York, N.Y.)*, 326(5950):234–6, October 2009.
- [Fur96] John L. Furlani. The Environment Modules package: <http://modules.sourceforge.net/>, 1996.
- [Gar07] Simson L Garfinkel. An Evaluation of Amazon’s Grid Computing Services: EC2, S3 and SQS. Technical report, Harvard University, 2007.
- [Gen05] Robert Gentleman. Reproducible research: a bioinformatics case study. *Statistical applications in genetics and molecular biology*, 4, January 2005.
- [Gen11a] GenoLogics. GenoLogics LIMS: <http://www.genologics.com/>, 2011.
- [Gen11b] Genomespace.org. GenomeSpace, bringing bioinformatics tools together: <http://genomespace.org>, 2011.
- [Gle11] Travis C. Glenn. Field guide to next-generation DNA sequencers. *Molecular Ecology Resources*, May 2011.
- [GNTG10] Jeremy Goecks, Anton Nekrutenko, James Taylor, and The Galaxy Team. Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences. *Genome Biology*, 11(8):R86, 2010.

BIBLIOGRAPHY

- [GSMG11] Dov Greenbaum, Andrea Sboner, Ximeng Jasmine Mu, and Mark Gerstein. Genomics and Privacy: Implications of the New Reality of Closed Data for the Field. *PLoS Computational Biology*, 7(12):e1002278, December 2011.
- [GTBK11] Angela Goncalves, Andrew Tikhonov, Alvis Brazma, and Misha Kapushesky. A pipeline for RNA-seq data processing and quality assessment. *Bioinformatics (Oxford, England)*, January 2011.
- [Gui] Roman Valls Guimera. How to install python modules with VirtualEnv... on UPPMAX.
- [Gui11a] Roman Valls Guimera. Galaxy on UPPMAX, simplified: <http://blogs.nopcode.org/brainstorm/2011/08/22/galaxy-on-uppmax-simplified/>, 2011.
- [Gui11b] Roman Valls Guimera. On the Galaxy API: <http://blogs.nopcode.org/brainstorm/2011/07/11/galaxy-community-conference-2011/>, 2011.
- [GWCY11] Anthony Goddard, Nathan Wilson, Phil Cryer, and Grant Yamashita. Data hosting infrastructure for primary biodiversity data. *BMC Bioinformatics*, 12(Suppl 15):S5, 2011.
- [HLCB11] Markus Hsi-Yang Fritz, Rasko Leinonen, Guy Cochrane, and Ewan Birney. Efficient storage of high throughput sequencing data using reference-based compression. *Genome Research*, January 2011.
- [HMRH11] Brandon Heller, Eli Marschner, Evan Rosenfeld, and Jeffrey Heer. Visualizing collaboration and influence in the open-source software community. In *Proceeding of the 8th working conference on Mining software repositories - MSR '11*, page 223, New York, New York, USA, 2011. ACM Press.
- [HWS⁺06] Duncan Hull, Katy Wolstencroft, Robert Stevens, Carole Goble, Mathew R Pocock, Peter Li, and Tom Oinn. Taverna: a tool for building and running workflows of services. *Nucleic Acids Research*, 34(Web Server issue):W729–32, July 2006.
- [Ioa05] John P A Ioannidis. Why most published research findings are false. *PLoS Medicine*, 2(8):e124, August 2005.
- [Kan03] Luke Kanies. Puppet: <http://puppetlabs.com/>, 2003.
- [Kar11] Konstantinos Karasavvas. Enacting Taverna workflows in Galaxy, 2011.
- [KCW⁺07] Lukas Käll, Jesse D Canterbury, Jason Weston, William Stafford Noble, and Michael J MacCoss. Semi-supervised learning for peptide identification from shotgun proteomics datasets. *Nature Methods*, 4(11):923–5, November 2007.
- [Kra11] Per Kraulis. slog: simple lab logging system: <https://github.com/pekrau/slog>, 2011.
- [KWV11] Olli Kallioniemi, Lodewyk Wessels, and Alfonso Valencia. On the organization of bioinformatics core services in biology-based research institutes. *Bioinformatics (Oxford, England)*, 27(10):1345, May 2011.

BIBLIOGRAPHY

- [LGG11] Burkhard Linke, Robert Giegerich, and Alexander Goesmann. Conveyor: a workflow engine for bioinformatic analyses. *Bioinformatics (Oxford, England)*, January 2011.
- [LHW⁺09] Heng Li, Bob Handsaker, Alec Wysoker, Tim Fennell, Jue Ruan, Nils Homer, Gabor Marth, Goncalo Abecasis, and Richard Durbin. The Sequence Alignment/Map format and SAMtools. *Bioinformatics (Oxford, England)*, 25(16):2078–9, August 2009.
- [Lim11] Limswiki. LimsWiki, a LIMS vendors list: http://limswiki.org/index.php/Main_Page, 2011.
- [LOSK08] Peter Li, Tom Oinn, Stian Soiland, and Douglas B Kell. Automated manipulation of systems biology models using libSBML within Taverna workflows. *Bioinformatics (Oxford, England)*, 24(2):287–9, January 2008.
- [LR09] Fran Lewitter and Michael Rebhan. Establishing a successful bioinformatics core facility team. *PLoS Computational Biology*, 5(6):e1000368, June 2009.
- [LRRS09] Fran Lewitter, Michael Rebhan, Brent Richter, and David Sexton. The need for centralization of computational biology resources. *PLoS Computational Biology*, 5(6):e1000372, June 2009.
- [Mer10a] Zeeya Merali. Computational science: ...Error. *Nature*, 467(7317):775–7, October 2010.
- [Mer10b] Stephan Mertens. Spotlight on plagiarism. *Deutsches Ärzteblatt international*, 107(49):863–5, December 2010.
- [Mic09] Sun Microsystems. Jenkins, an extendable open source continuous integration server: <http://jenkins-ci.org/>, 2009.
- [MJ09] Brian Warner Mitchell and Dustin J. Buildbot, a python system to automate the compile/test cycle: <http://trac.buildbot.net/>, 2009.
- [MK12] Daniel MacLean and Sophien Kamoun. Big data in small places. *Nature Biotechnology*, 30(1):33–34, January 2012.
- [Nat10] Nature. Plagiarism pinioned. *Nature*, 466(7303):159–60, July 2010.
- [NBL⁺10] Sarah B Ng, Kati J Buckingham, Choli Lee, Abigail W Bigham, Holly K Tabor, Karin M Dent, Chad D Huff, Paul T Shannon, Ethylin Wang Jabs, Deborah A Nickerson, Jay Shendure, and Michael J Bamshad. Exome sequencing identifies the cause of a mendelian disorder. *Nature genetics*, 42(1):30–5, January 2010.
- [Ops09] Opscode. Opscode chef: <http://www.opscode.com/>, 2009.
- [O’S09] Bryan O’Sullivan. Making sense of revision-control systems. *Communications of the ACM*, 52(9):56, September 2009.
- [PBK10] Julian Parkhill, Ewan Birney, and Paul Kersey. Genomic information infrastructure after the deluge. *Genome biology*, 11(7):402, January 2010.
- [Pen11] R. D. Peng. Reproducible Research in Computational Science. *Science*, 334(6060):1226–1227, December 2011.

BIBLIOGRAPHY

- [PLS⁺11] Laurence D. Parnell, Pierre Lindenbaum, Khader Shameer, Giovanni Marco Dall’Olio, Daniel C. Swan, Lars Juhl Jensen, Simon J. Cockell, Brent S. Pedersen, Mary E. Mangan, Christopher A. Miller, and Istvan Albert. BioStar: An Online Question & Answer Resource for the Bioinformatics Community. *PLoS Computational Biology*, 7(10):e1002216, October 2011.
- [PVP⁺12] Swetansu Pattnaik, Srividya Vaidyanathan, Durgad G. Pooja, Sa Deepak, and Binay Panda. Customisation of the Exome Data Analysis Pipeline Using a Combinatorial Approach. *PLoS ONE*, 7(1):e30080, January 2012.
- [RLG⁺06] Michael Reich, Ted Liefeld, Joshua Gould, Jim Lerner, Pablo Tamayo, and Jill P Mesirov. GenePattern 2.0. *Nature Genetics*, 38(5):500–1, May 2006.
- [RMH⁺10] Arcot Rajasekar, Reagan Moore, Chien-Yi Hou, Christopher A. Lee, Richard Marciano, Antoine de Torcy, Michael Wan, Wayne Schroeder, Sheau-Yen Chen, Lucas Gilbert, Paul Tooby, and Bing Zhu. iRODS Primer: Integrated Rule-Oriented Data System. *Synthesis Lectures on Information Concepts, Retrieval, and Services*, 2(1):1–143, January 2010.
- [RPP⁺11] K. Rother, W. Potrzebowski, T. Puton, M. Rother, E. Wywiał, and J. M. Bujnicki. A toolbox for developing bioinformatics software. *Briefings in Bioinformatics*, July 2011.
- [RS09] Brent G Richter and David P Sexton. Managing and analyzing next-generation sequence data. *PLoS Computational Biology*, 5(6):e1000369, June 2009.
- [RSBM⁺10] Philippe Rocca-Serra, Marco Brandizi, Eamonn Maguire, Nataliya Sklyar, Chris Taylor, Kimberly Begley, Dawn Field, Stephen Harris, Winston Hide, Oliver Hofmann, Steffen Neumann, Peter Sterk, Weida Tong, and Susanna-Assunta Sansone. ISA software suite: supporting standards-compliant experimental annotation and enabling curation at the community level. *Bioinformatics (Oxford, England)*, 26(18):2354–6, September 2010.
- [RWC⁺10] Jingyuan Ren, Nadya Williams, Luca Clementi, Sriram Krishnan, and Wilfred W Li. Opal web services for biomedical applications. *Nucleic Acids Research*, 38 Suppl:W724–31, July 2010.
- [Sam02] Peter Samuel. Graft - a package management utility: <http://peters.gormand.com.au/Home/tools/graft/graft-html>, 2002.
- [Sis11] Jordan Sissel. FPM: <https://github.com/jordansissel/fpm.git>, 2011.
- [SKA⁺10] Henrik Stranneheim, Max Käller, Tobias Allander, Björn Andersson, Lars Arvestad, and Joakim Lundberg. Classification of DNA sequences using Bloom filters. *Bioinformatics (Oxford, England)*, 26(13):1595–600, July 2010.
- [Ste96] Lincoln Stein. How Perl saved the Human Genome Project: http://www.bioperl.org/wiki/How_Perl_saved_human_genome, 1996.
- [Ste10] Lincoln D Stein. The case for cloud computing in genome informatics. *Genome Biology*, 11(5):207, May 2010.
- [Ste11] Sam Stephenson. rbenv - Simple Ruby Version Management: <https://github.com/sstephenson/rbenv>, 2011.

BIBLIOGRAPHY

- [Sto10] Michael Stonebraker. SQL databases v. NoSQL databases. *Communications of the ACM*, 53(4):10, April 2010.
- [TD10] Hong-Linh Truong and Schahram Dustdar. Cloud computing for small research groups in computational science and engineering: current status and outlook. *Computing*, 91(1):75–91, September 2010.
- [Tom04] Ryan Tomayko. How I Explained REST to My Wife: <http://tomayko.com/writings/rest-to-my-wife>, 2004.
- [TRHD07] Peter Troger, Hrabri Rajic, Andreas Haas, and Piotr Domagalski. Standardization of an API for Distributed Resource Management Systems. In *Seventh IEEE International Symposium on Cluster Computing and the Grid (CCGrid '07)*, pages 619–626. IEEE, May 2007.
- [vR98] Guido van Rossum. Glue It All Together With Python, 1998.
- [Wik05] Wikipedia. The distributed Lustre filesystem: [http://en.wikipedia.org/wiki/Lustre_\(file_system\)](http://en.wikipedia.org/wiki/Lustre_(file_system)), 2005.
- [Wik11a] Wikipedia. http://en.wikipedia.org/wiki/Beowulf_cluster, 2011.
- [Wik11b] Wikipedia. http://en.wikipedia.org/wiki/Buffer_overflow, 2011.
- [Wik11c] Wikipedia. http://en.wikipedia.org/wiki/Laboratory_information_management_system, 2011.
- [Wik11d] Wikipedia. SetUID: <http://en.wikipedia.org/wiki/Setuid>, 2011.
- [Wik11e] Wikipedia. Technical debt: http://en.wikipedia.org/wiki/Technical_debt, 2011.
- [WSP⁺07] Michael C Wendl, Scott Smith, Craig S Pohl, David J Dooling, Asif T Chinwalla, Kevin Crouse, Todd Hepler, Shin Leong, Lynn Carmichael, Mike Nhan, Benjamin J Oberkfell, Elaine R Mardis, LaDeana W Hillier, and Richard K Wilson. Design and implementation of a generalized laboratory data model. *BMC Bioinformatics*, 8:362, January 2007.
- [WTS10] WTSI. Sanger SequenceScape: <https://github.com/sanger/sequencescape>, 2010.
- [Xtr02] Juan Xtraeme. XBPS - The X Binary Package System: <http://code.google.com/p/xbps/>, 2002.

TRITA-CSC-E 2012:020
ISRN-KTH/CSC/E--12/020-SE
ISSN-1653-5715