Chipster tool development and server administration

Aleksi Kallio, CSC – IT Center for Science

December 14th 2012, CHPC, Cape Town

Technical introduction to Chipster

Technical introduction

- Chipster is a graphical application for data analysis, with server backend
- Oriented for integration of existing tools, databases and visualisations
- Easily modifiable, extendable etc.
- User oriented approach to everything
- For more information, the best reference is Technical manual
  - https://github.com/chipster/chipster/wiki

Architecture

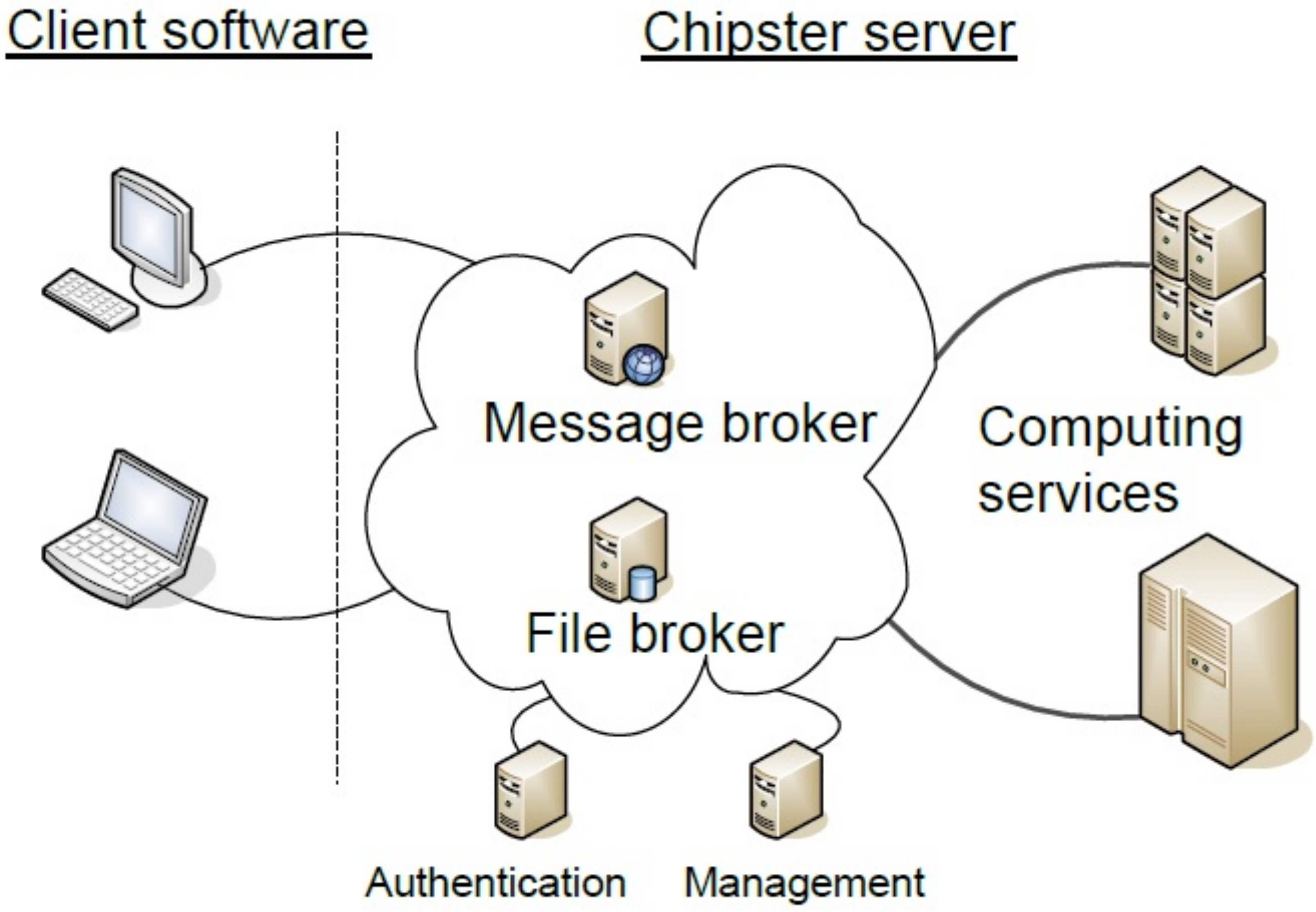
- Under the hood, the system is built on message oriented architecture
- Components communicate by broadcasting messages
- Components are not directly aware of each other => loosely coupled communication
- Message broker takes care of moving messages around

Major components

- Client application (GUI)
  - Thick client = most of the logic is in the client
- Computer service
- Authentication service
- Message broker (ActiveMQ)
- File broker (Jetty)

Architecture

Message passing, thick client



Technologies used

- System itself is 100% Java, incl. the client
- File broker and webstart server based on Jetty www-server
- Message broker is ActiveMQ (JMS)
- Admin console stores log data to H2 database (SQL) and offers H2 console web interface
- Server uses Java Service Wrapper to wrap Java into Linux/etc daemons
- All of the previous are integrated into Java code and mostly you don’t need to care about them
- Tools are R, C, Perl, Python, Java...

Chipster development

- Open source project with various contributing parties
- Relies heavily on other open source projects
- Core development team at CSC, Finland
- Contributors in various universities and companies in Finland, Netherlands, Germany, Australia...
- Happy to get new people contributing ideas, code, bug reports, documentation, etc.!

Adding new analysis tools in Chipster

What happens before client is able to draw tool GUI?

- Tool script is sitting in **modules** directory of the server (compute service)
- Client is started and it requests tool descriptions
- Server checks the script file for changes and sends all descriptions
- Client parses the descriptions
- When tool is selected, the parsed description is used to generate the parameter panel
- Some parameters depend on data, so GUI might look different based on which dataset is selected

How changes become visible?

- Every time server uses tool script, it checks for changes
- Client generates the skeleton of GUI at startup
- Changing tool code => no restarts (common)
- Adding, removing tools or changing parameters of tools => client restart, no server restart (not common)
- Adding, removing complete module => client and server restart (extremely rare)
- Integrating tools to Chipster is a streamlined process

SADL tool descriptions

```
TOOL concat.R: "Concatenate tool" (...)
INPUT file1.txt: "First input" TYPE GENERIC (...)
INPUT file2.txt: "Second input" TYPE GENERIC (...)
OUTPUT concatenated.txt: "Concatenated file" (...)
```

SADL tool descriptions

```
TOOL util-test.R: "Test tool" (...)
INPUT microarray{...}.tsv: "Raw data files" TYPE CDNA (...)
INPUT META phenodata.tsv: "Experiment description" TYPE GENERIC (...)
OUTPUT result.txt: "Result file" (...)
OUTPUT OPTIONAL warnings.txt: "Warning file" (...)
PARAMETER value1: "The first value" TYPE INTEGER
  FROM 0 TO 200 DEFAULT 10 (...)
PARAMETER OPTIONAL value2: "The second value" TYPE DECIMAL
  FROM 0 TO 200 DEFAULT 20.2 (...)
PARAMETER method: "Method" TYPE
  [linear: "Linear scale", logarithmic: "Logarithmic scale"]
  DEFAULT logarithmic (...)
PARAMETER genename: "Gene name" TYPE STRING DEFAULT at_1234 (...)
PARAMETER key: "Key column" TYPE COLUMN_SEL (...)
```

What lives inside compute service

- Service has several **runtimes**
  - Example: R 2.12 is one runtime
  - Defined in runtimes.xml
- Each runtime has one **analysis handler**
  - Defines what kind of tools the runtime is capable of running
- Service has also several **modules**
- Module contains **tools**, which are grouped to **categories**
- Configuration must match across nodes
- Tools and runtimes can be disabled per node

Recap

- **Runtime:** dynamic object that actually runs the tools (e.g. R interpreter)
- **Analysis handler:** Protocol to handle certain styles of tools (e.g. R scripts)
- **Module:** collection of tools for certain area (e.g. NGS data analysis)
- **Tools:** something user can run
- **Category:** grouping of tools, only to draw nicer GUI
- So tools are in two hierarchies: runtimes/ analysis handlers for running them and modules/categories for showing them

Integrating R/Bioconductor scripts

- The most advanced analysis handlers and runtimes are for R
- R interpreters are pooled, so that job startup time is minimized
  - R is a good general purpose wrapper language in Chipster
- There is a small but growing collection of common useful functions for using R/ Bioconductor with Chipster
- You can output special string CHIPSTER-NOTE to send formatted message to user

Integrating R/Bioconductor scripts

- Modifying tool code step by step
  - Change script
  - Test that it works
- Adding tool step by step
  - Add to module.xml
  - Write the script
  - (Re)start client
  - Test that it works

Integrating command line tools

- It is possible to directly integrate command line tools by writing a bare SADL description file and attaching it with shell analysis handler
  - Parameter parsing can be awkward
  - No pre or post processing
  - Dumping command line tools directly often not user friendly
- Recommendation: wrap command line tools with scripting language
  - R, Java or BeanShell directly supported

Other languages?

- What about Python, Perl, Ruby, Python, Scala, C...
- Options, from easy to less easy:
  - Use R to wrap your script
  - Use shell handler and wrap inside your own script (parse arguments)
  - Ask us to implement new analysis handler
  - Implement new analysis handler
- Later options of course better in long run

Implementing new analysis handlers

- Using the Java API, it is possible to:
  - Implement your own tool types
  - Implement your own runtimes
  - Integrate whole tool repositories
- Example: Embster = EMBOSS+others
  - No conversions are needed, handler reads EMBOSS ACD files directly

Running tools in your workstation

- Typically everything is run on server
- To run locally, options are:
  - Export data, run, import data
  - Deploy compute service to your workstation
  - Use Java API to implement local tool
    - Not recommended, but has been done for NGS preprocessing
- Local execute: if people need this, can be easily implemented

EXERCISES

Setting up Chipster server

Setting up Chipster server

- Two major options:
  - Recommended: Chipster virtual machine (VM)
  - Not so recommended: Clean installation to Linux, Unix or Mac OS X
- Other options:
  - Hybrid, install your own Debian flavor Linux that is compatible with Chipster VM and copy things over
  - Don't try this at home: Clean install to Windows...

What is Chipster virtual machine?

- Chipster server + all tools + all databases + Ubuntu Linux = Chipster virtual machine (VM)
- Supports all major virtualisation platforms
  - KVM, VMware, VirtualBox
- Recommended platforms:
  - Windows: VMware or VirtualBox
  - Mac OS X: VirtualBox
  - Linux: VirtualBox
  - Clusters: KVM

What is Chipster virtual machine?

- Chipster VM is available at http://chipster.github.io/chipster/
- It is sizeable: around 200 gigabytes
  - Contains annotation data, reference genomes, various databases...
- Why is it so huge?
  - Producing new virtual machine every ~two weeks is a complicated and heavy process
  - Can produce only limited selection of VM's
  - Currently producing VM that has it all

Will it be huge in future also?

- Currently the first download is huge, after that you can use update mechanism to get only things that have changed
- Work on CernVM-FS

DEMO

Clean install on Linux

- Chipster installation is easy, unpack and run configure
- Analysis tools need more work
- If your Linux is similar to Ubuntu, you can follow our virtual machine install script and installation is easy
  - If you are close enough, you can just copy binaries over and it is very easy
- Otherwise need to find out how different applications can be installed to your environment
- Genomes and databases are easy, because they are just data

Keeping installation up to date

Keeping VM installation up to date

- When starting the VM, you should update the operating system (using aptitude or apt-get)
- It is recommended to have periodical checks for operating system updates to keep it secure
- Chipster you need to update only when new functionality is needed
  - If there are security issues, patches are announced via the mailing list
- Chipster update happens automatically when you call **update.sh** script

Keeping clean installation up to date

- If you are installing from scratch, you probably know how to keep operating system up to date
- Chipster update tool is not supported outside of VM
- However **update-exec.sh** script can be used as a specification for things that need to be updated between versions
- So to update, look at the script and either make it runnable in your environment orrepeat same steps manually

Production level server installation

About production systems

- Every production environment is different
- Following will be based on our experience
    - We have been developing Chipster for 10 years and running it in production for 7 years
- Our environment: supercomputing center that also takes care of the national university network backbone
    - History of environments: large Sun Solaris machine, physical HPC Linux cluster, OpenNebula/KVM cluster, OpenStack
    - Other nodes: physical Linux boxes, virtual VMware boxes

Distributed compute nodes

- Chipster compute nodes are following the worker pattern
- You can start and stop them freely
  - When node is killed, you loose jobs that were running there, but nothing else
- There can be many and they can be located on different servers and behind firewalls
- Especially for NGS jobs, it is recommended to have more nodes doing the computation
  - Typically memory is the limiting factor
  - Can be controlled by setting maximum job count per node

Monitoring

- It is not good if your users need to tell you that your system is down
- Nagios (or similar) monitoring system can be used to monitor Chipster server
  - You get notified when system is down
  - Notifications via email, SMS…
  - To prevent false alarms, you can define your system topology
  - E.g.: if network is down, don’t complain about server
- Nagios can track monthly availability and similar statistics

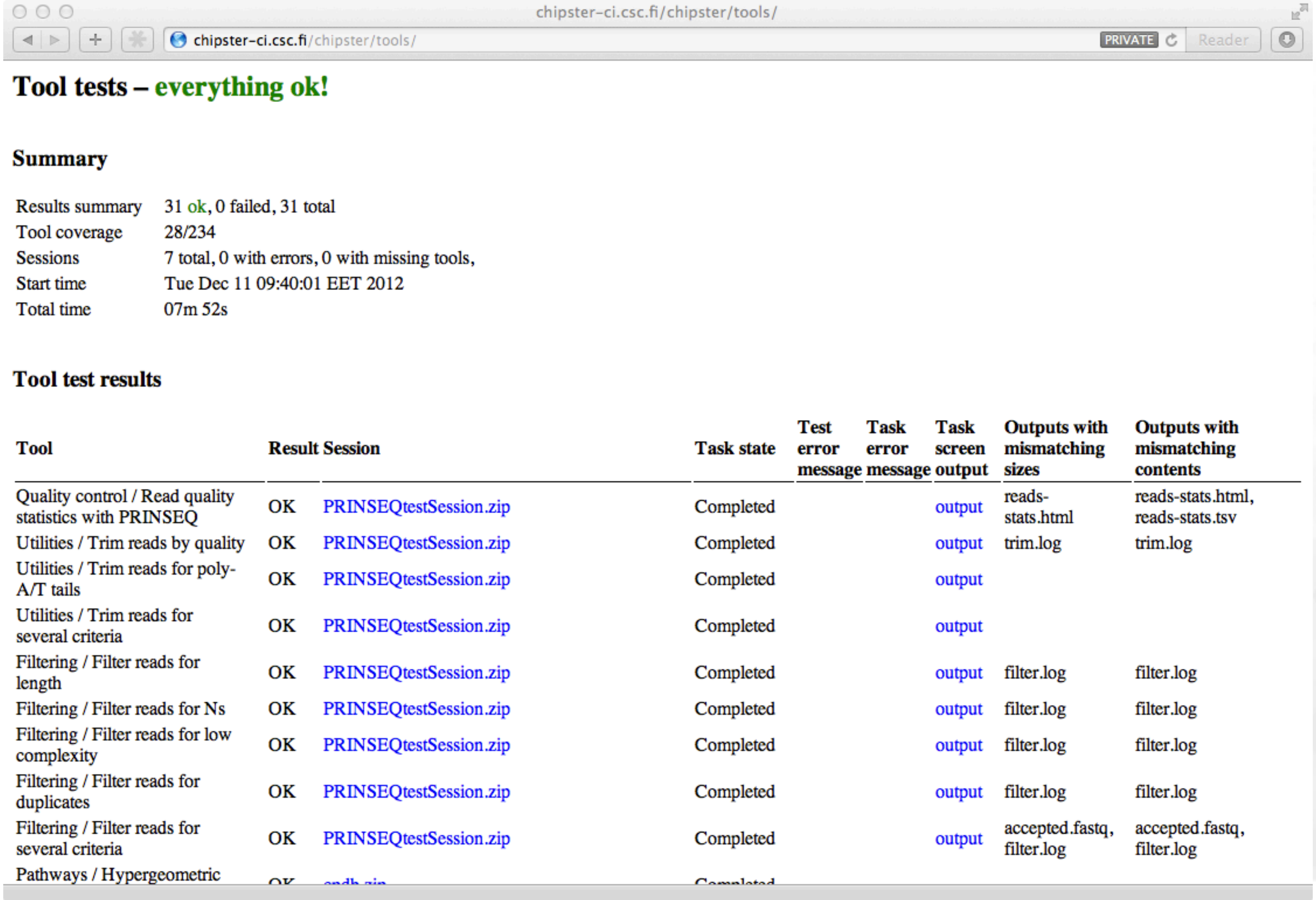
Monitoring

- To implement Nagios monitoring, you can use command line switch **nagios-check**
  - Prints Nagios compatible output to stdout/stderr
  - Chipster client needs to exist on Nagios host
- Or use testrunner, described next

Continuous testing

- It is easy to write software that works. It is a lot harder to write software that works 24x7.
- Chipster testrunner can be used to constantly test the system
- Creating test cases is very easy: just save a session, place it in testrunner folder and testrunner tries to repeat it
- Test run produces a test report
- You don't want to constantly check the reports, so they can be monitored with Nagios

Testrunner report


chipster-ci.csc.fi/chipster/tools/
chipster-ci.csc.fi/chipster/tools/
PRIVATE
Reader

Tool tests – everything ok!

Summary

Results summary	31 ok, 0 failed, 31 total
Tool coverage	28/234
Sessions	7 total, 0 with errors, 0 with missing tools,
Start time	Tue Dec 11 09:40:01 EET 2012
Total time	07m 52s

Tool test results

Tool	Result	Session	Task state	Test error message	Task error message	Task screen output	Outputs with mismatching sizes	Outputs with mismatching contents
Quality control / Read quality statistics with PRINSEQ	OK	PRINSEQtestSession.zip	Completed			output	reads-stats.html	reads-stats.html, reads-stats.tsv
Utilities / Trim reads by quality	OK	PRINSEQtestSession.zip	Completed			output	trim.log	trim.log
Utilities / Trim reads for poly-A/T tails	OK	PRINSEQtestSession.zip	Completed			output		
Utilities / Trim reads for several criteria	OK	PRINSEQtestSession.zip	Completed			output		
Filtering / Filter reads for length	OK	PRINSEQtestSession.zip	Completed			output	filter.log	filter.log
Filtering / Filter reads for Ns	OK	PRINSEQtestSession.zip	Completed			output	filter.log	filter.log
Filtering / Filter reads for low complexity	OK	PRINSEQtestSession.zip	Completed			output	filter.log	filter.log
Filtering / Filter reads for duplicates	OK	PRINSEQtestSession.zip	Completed			output	filter.log	filter.log
Filtering / Filter reads for several criteria	OK	PRINSEQtestSession.zip	Completed			output	accepted.fastq, filter.log	accepted.fastq, filter.log
Pathways / Hypergeometric	OK	cpdb.zip	Completed					

Continuous testing

- You can have multiple test suites
- Possible example setup:
  - Complete suite, run every 2 hours, Nagios monitored with low criticality
  - Minimal suite, run every 10 minutes, Nagios monitored with high criticality
- Nagios monitoring of testrunner reports is standard HTTP monitoring
  - No need to have Chipster client on Nagios host

Other production level topics

- Manager / admin web console
  - SQL database that collects central log information, web query interface
  - Useful for statistics, debugging, usage monitoring, etc.
- loghost
  - Synchronising logs to a separate and highly secure server
  - Important ones are activemq/data/activemq.log, fileserver/logs/chipster.log and auth/logs/chipster.log

Other production level topics

- Authentication
  - Authentication system has Java and JAAS APIs
  - JAAS is the Java authentication standard, giving support to various providers like LDAP
- SSL
  - Message broker supports SSL
  - File broker supports SSL (in 3.0)
  - Not enabled by default, but fairly straightforward to do it
  - Secure communications should typically be used