Robust architectural patterns for bioinformatics: experiences with Chipster

Aleksi Kallio, Taavi Hupponen, Petri Klemelä, Jarno Tuimala, Eija Korpelainen
CSC - IT Center for Science Ltd. (contact address: aleksi.kallio@csc.fi)

Open source project homepage: http://chipster.sourceforge.net
Source code: http://chipster.sourceforge.net/sources.shtml

Chipster (http://chipster.csc.fi) is data analysis platform which offers an intuitive graphical user interface to a comprehensive collection of DNA microarray data analysis methods, such as those developed in the R/Bioconductor project. As Chipster was developed for large scale, national use, the concerns for usable, robust and dependable technology had to be addressed carefully.

Based on our previous experience from developing and maintaining distributed systems we decided to opt for component based asynchronous architecture as a base paradigm. The Chipster environment consists of components that use message oriented middleware (Java JMS) for communication. The most important characteristic of the Chipster architecture is how distributed state is managed. Instead of complex patterns such as 2-phase commits we have designed components to be independent and to employ best effort service. This means that there are no hard state dependencies between the components and, for example, they can be restarted independently, making the system very adaptable to changes in the physical setup. Chipster allows compute node configuration to be changed at runtime and we have extended it to allow also data brokers to be configured in the same way. Hardware problems, smaller software problems and configuration changes can be taken care without affecting connected users.

For improving the perceived performance of the system, the most important step is to closely monitor performance. The message oriented middleware layer is used to communicate status messages and they are collected to a centralized database. We have implemented a manager tool that enables real time inspection of the status of the system and. By integration into Nagios monitoring platform we get real time alerts at system wide shortages.

Finally, one can never make the system bullet proof. However, robustness can be improved by putting effort into error management. For example, it is important to produce understandable information for users when the system not completely responsive. The benefit of a thick client architecture is that we can implement error management on the client layer, so that network or server side lags do not directly hinder user experience. Lately we have concentrated on improving client side robustness, for example by isolating visualization processes that can consume large amounts of CPU resources.

Taken together, after 2 years of public service on top of the Chipster platform we feel that our efforts with robust design and implementation have paid off. We have been able to concentrate on further development instead of nurturing an ill-behaving service. We feel that focusing on robust architectural patterns and their sharing will be beneficial for bioinformatics software developers.