5 Parallel incomplete ICA

Functional magnetic resonance imaging (fMRI, [34,43]) based on the blood oxygen level dependent contrast (BOLD) nowadays is one of the main technologies in human brain research. Using spatial independent component analysis (ICA) to interpret fMRI data (see e.g. [39] for an application) works well, as the functional segregation of the brain closely matches the requirement of statistical spatial independence. The advantage of ICA for the analysis of fMRI data is that it is a model free technique so that it is not necessary to know in advance how the brain will react to the stimulus that is presented to the subject.

Basically, spatial ICA separates the fMRI data set $\mathbf{X}$ into statistically independent sources $\mathbf{S}$ (in the case of fMRI they represent maps of activation in the brain) and their time courses, which are contained in the so called mixing matrix $\mathbf{A}$:

$$\mathbf{X} = \mathbf{A}\mathbf{S} \tag{5.1}$$

A severe problem in the case of fMRI is the mass of data that has to be analysed manually to find the interesting components, as each fMRI session will typically yield hundreds of different components. Also, the calculations tend to be time consuming – a single processor desktop computer can need more than one hour to do a complete ICA of a one-subject data set of 400 normal-sized fMRI Images with the *Matlab* implementation of *FastICA* [24].

5.1 Theory

One way to deal with the mentioned problem is the use of a parallel algorithm on multiple processors. Constructing a parallel algorithms usually is not a simple matter, however in the case of ICA applied to fMRI the way to do this is rather straightforward (apart from the idea to create a parallel and fast ICA algorithm – this to the knowledge of the author has not been accomplished so far). In applied incomplete ICA fortunately most parts of the process do not depend on the results the other parts, so that they can be executed independently.

A typical incomplete ICA session consists of multiple ICAs with different dimension settings and the corresponding reliability tests for these ICAs. In the following sections I will give a more detailed description on how to compute these parts parallel.

5.1.1 Parallel Reliability Testing

As argued in chapter 2, testing the reliability of an ICA can be important in deciding whether or not an ICA result is valid. This test always consist of running the ICA algorithm multiple times and then comparing the results. As these runs are not depending on the results of each other, they can be calculated parallel without any problems (figure 5.1). Only the comparison of the different runs can not be done without receiving all of the results.

5.1.2 Incomplete ICA with varying dimensions

Using multiple, incomplete ICA runs with increasing dimensions that will cluster the activation maps based on their similarity of their time courses in the data set, it is possible to get a complete view of the sources that are contained in a dataset. This procedure has various advantages:

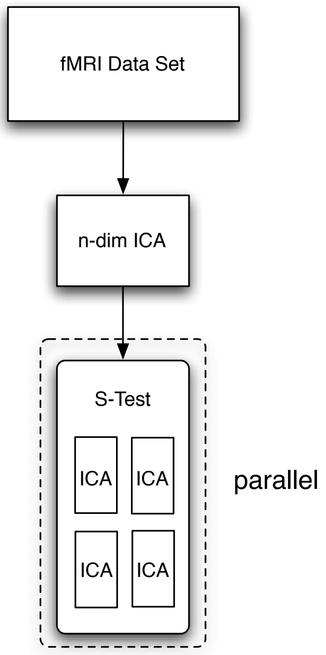

Figure 5.1: Diagram of parallel stability-/reliability-testing ("S-Test") with ICA. The dotted boxes represent the parts of the program that can be executed parallel.

- The researcher can start with a few clusters of a low-dimensional ICA and investigate the separate, localised activation patterns in a high-dimensional ICA that form this cluster. This is especially useful in finding networks of active regions in the brain and studying their behaviour.

- The number of sources contained in the data set usually is difficult to estimate. Using multiple ICAs with different settings for the dimension of the sources it should become apparent from the results at which dimension the best results can be found.

- The overview can allow the researcher to gain a deeper understanding of the way the ICA is analysing his data set and of the results of the ICA.

The ICA runs do not depend on the results of each other. It is therefore possible to do a parallel calculation like in the case of the reliability testing. At the same time the

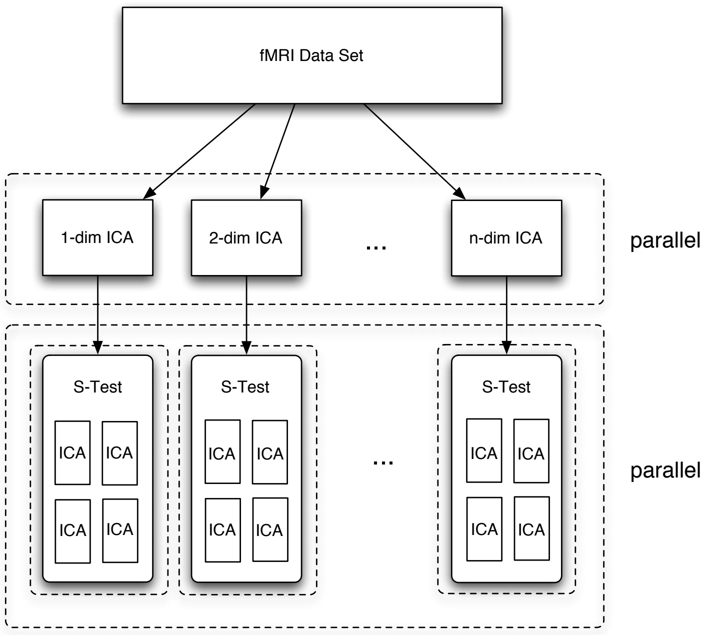

Figure 5.2: Diagram of parallel clustering with incomplete ICA. The dotted boxes represent the parts of the program that can be executed parallel.

reliability-testing can be done parallel, so the case of applied incomplete ICA can be called massive parallel problem (see figure 5.2).

The most time-consuming parts of this method are the ICA runs. Comparing the resulting estimated components for their stability happens much faster – on a modern PC this can be done in seconds, while the ICA runs will take minutes to hours. As all the ICA runs can be done parallel, the gain from running the inICA parallel is almost direct proportional to the number and speed of processors that can be used.

5.2 Implementation

The implementation of a parallel version of incomplete ICA had to meet some important requirements to be easy to implement on various sites:

1. no additional licence costs

2. handling of high-volume data sets

3. apart from MATLAB [36] or GNU Octave [12] only standard software on unix-like systems should be used.

A common way in parallel computing is to utilise message-based infrastructures like the Message-Passing Interface Standard (MPI) [16] or the Parallel Virtual Machine (PVM)[1] [57]. However, these methods have limitations that make them difficult to apply in this case:

- MPI and PVM are not standard software on unix-like systems. While they are commonly used in universities, they require additional software to be used with MATLAB or GNU Octave.

- fMRI data analysis generates huge amounts of data that is unsuited to be transferred with a messaging system.

The last point is a very serious problem in message-based infrastructures because they are usually designed only for relatively small messages of few kB. At the same time, however, the handling of huge data blocks over the network is a normal task for a modern network file system. It seems therefore to be a good idea to use this advantage of the network file system by a simple server-client-system, that also uses the file system as message path:

- The server process generates matlab/octave-files that will be processed by the clients. These files are stored in a common directory that is monitored by the clients. The server is also the interface to the user.

- The intelligent clients monitor the common directory for work files from the server process. If a client is idle it will process the next work file and save the generated data in the common directory.

[1]http://www.csm.ornl.gov/pvm/pvm_home.html

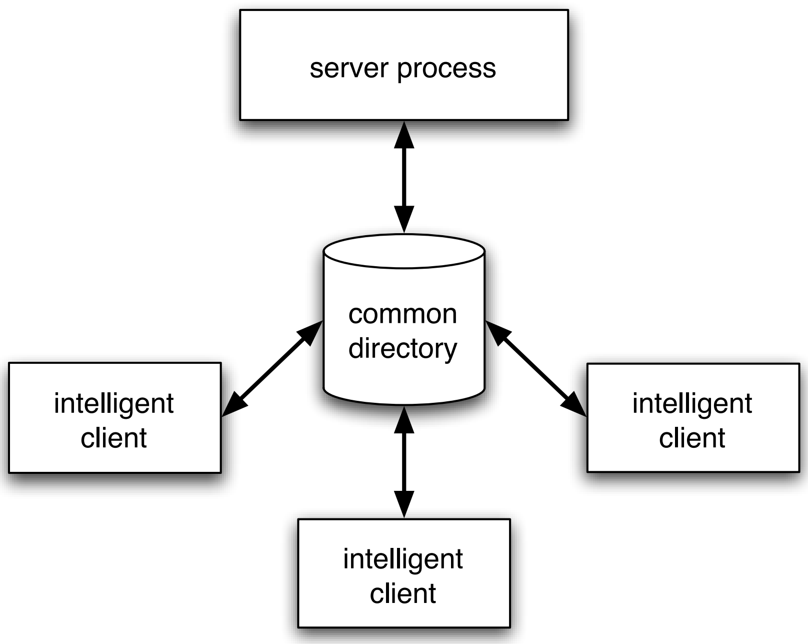

Figure 5.3: A concept of a server-client-system with intelligent clients. Data and work can be interchanged via a common directory.

Figure 5.3 shows a sketch this concept.

This method has some advantages over message-based procedures:

- Deadlocks due to message-transfer-problems are not possible (i.e. client and server waiting for a message from each other).
- It is possible to automatically balance the load on the clients if the clients support this.
- The file-system is used to store and retrieve the data, therefore handling huge data sets should work without problems.

The disadvantages are:

- Messaging "abusing" the file system is very slow as it depends on the clients and the server monitoring the common directory for changes. The clients will not be able to monitor the directory every second as this imposes stress on the file-server and the network infrastructure. If the processing time of the tasks is in the same region as the obligatory pause before the clients will notice the task, a real message-transfer method like MPI will be much better suited for the task.

- Deadlocks due to errors in the job-scripts are possible. Care has to be taken that the server script takes account of time-outs.

5.2.1 Communication

The communication between the server process and the client processes happens via files that are created in the common directory. The different types of files are:

jxxxxx.m These are files that contain the jobs for the clients. These files are matlab or octave scripts and have an unique number xxxxx to identify them.

jxxxxx.mat Data-files from matlab or octave that contain the results from job jxxxxx.m.

jxxxxx.log A flag-file that is created if a client starts the job jxxxxx.m and contains the textual output from this job.

tmpxxxxx*.* Temporal files that will not be analysed.

It is also possible to include a public key security into this concept, by accompanying the job file with a signature-file jxxxxx.sig that allows an integrity check.

5.2.2 Server Process

The server process handles the interface to the user. It creates the jobs-scripts for the clients and analyses the resulting data. To accomplish this it has to monitor the common directory for the data-files with the results and also has to clean up the directory from the files that are no longer of use.

The server process also has to take care that deadlocks due to missing data sets do not happen in the clients.

5.2.3 Intelligent Clients

The clients have to monitor the common directory for new job-scripts that do not already have a corresponding flag-file. If a client finds a new job, he creates the corresponding flag-file jxxxxx.log where xxxxx is the number of the job. This flag-file detains the other clients from working on the same job. The job-script can save it's results in a data-file corresponding to the name of the job and can use temporal files (also marked with the name of the job to help in the clean-up).

As most of the work of the client is contained in the job-scripts, the client itself is very simple:

1. Monitor the common directory until a job-script without flag-file is found,
2. create flag-file for the job-script,
3. start octave or matlab to execute the job-script,
4. after the job has been finished, continue with step 1.

A simple form of the client was written as a python script of 56 lines. It is included in appendix B.

5.3 Example

To demonstrate the result of the parallel incomplete ICA method the data-set of a WCST experiment that was analysed in section 4.4 was used. Figure 5.4 gives small example of an incomplete ICA. Each column represents an (incomplete) ICA, the dimensions were 5, 10 and 15 (16 components are plotted because in the reliability testing one further component appeared. The border around each component represents the reliability of the component (darker is more reliable), tested using the correlation test and the jackknife method. The lines represent the overlap between the components (darker is more reliable, an overlap of more than 0.8 is plotted in bold blue).

This result is generated as a SVG-file[2] that can be viewed with any standard complying SVG viewer.

[2]scaleable vector graphic, http://www.w3.org/Graphics/SVG/

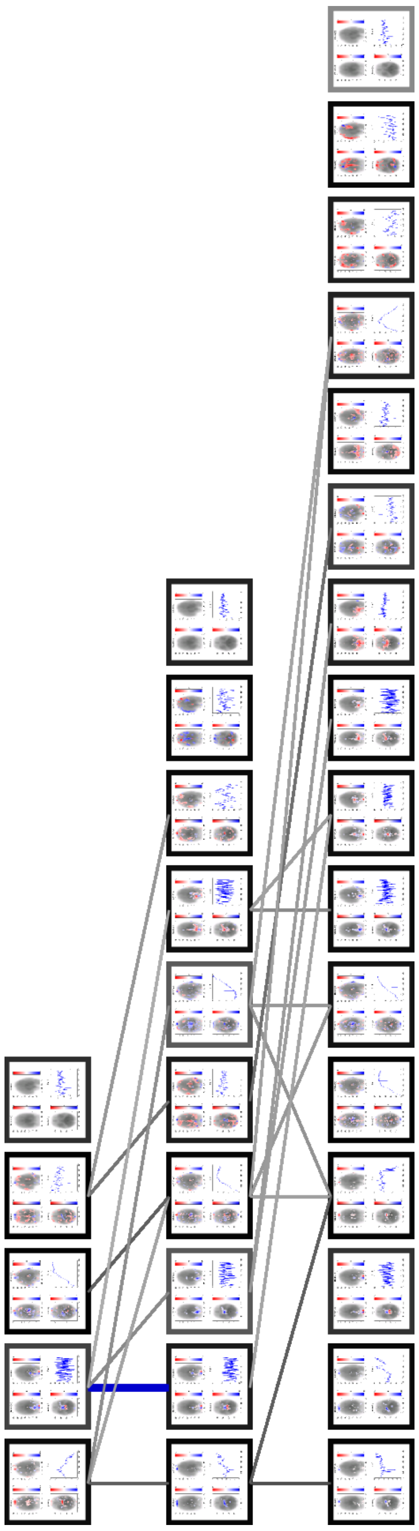

Figure 5.4: An example of an incomplete ICA. The lines mark the components that are related to each other, the colour of the border shows the stability of each component (darker is more reliable)