Design and development of a comprehensive data management platform for cytomics: cytomicsDB

Larios Vargas, E.

Chapter 5

Cluster Integration for Image processing and Pattern recognition

In this chapter is described how CytomicsDB supports the integration of cluster computing for the image analysis stage and how is performed the data processing and pattern recognition on MonetDB database. It contains also a detailed case study for two commonly used images analysis algorithms that have been adapted for efficient use on a computing cluster, and explore the effect on their performance.

This chapter is based on the following publication:

- E. Larios, D. van Es, K. Rietveld and F. J. Verbeek. **Cluster integration in CytomicsDB**. (Manuscript in preparation).

5.1 Introduction

In cytomics, the large-scale study of cell systems there is an urgent need to use high performance and parallel computing technologies due to the large volume of data managed in different type of experiments. One of the most used techniques in this area is High-Throughput screening (HTS) where thanks to the use of sophisticated microscopes a large set of cell images are acquired with taking pictures at a certain temporal interval. The resulting images can be studied individually, to observe cell characteristics such as morphology, or analysed as a time lapse series, to observe cell migration and motility for example.

It was mentioned in Chapter 1 that due to the complexity and variety of types of

data managed in HTS experiments it was necessary to design an automated workflow capable of handling the data properly in each stage of the experiment (LZY+12). The last 2 stages in HTS experiments: (1) *image analysis* and (2) *data analysis* receive as input a large volume of data, effectively it is physically impossible for a human examiner to go through every image and attempt to extract the required phenotypic measurements. Therefore, it is mandatory to use a computer-based environment for using image analysis techniques in order to ensure objective results.

In (YVDvdW09) and (YV12) are described two robust algorithms for image analysis customized for HTS studies. These algorithms were implemented using the Fiji software. This software is designed for the biologist and intended for a single user on a single machine using a sequential approach, interfacing through a GUI. For small tasks this is a proven setup, but it can prove impractical for high throughput experiments, with the large data sets often leading to long wait times and delays. A full analysis of one well plate can take two to three hours. Any possibility to speed up computation and decrease wait times is therefore highly desirable.

To support computation in the Life Sciences, at Leiden Institute of Advanced Computer Science (LIACS) has been built the Leiden Life Sciences Cluster (LLSC). This cluster consists of a fileserver, one user node and 24 worker nodes. This opens up possibilities to use a scalable and distributed environment for the Image Analysis stage. Additionally, the repository layer in the architecture of CytomicsDB relies on MonetDB database which provides a great platform for data analysis and visualization. The embedded MonetDB.R package serves as a powerful tool for data exploration and data mining.

In this chapter, we explore how CytomicsDB is integrated to the Cluster system, how the platform use the adapted image analysis algorithms on the LLSC and the performance of the resulting parallelized algorithms is also evaluated. Moreover, it is described the environment for data processing provided by MonetDB.

5.2 The Leiden Life Sciences Cluster

The Leiden Life Sciences Cluster (LLSC) is a computing cluster recently built at Leiden University. Its intended use is for research related to bioinformatics or other life sciences. All experiments conducted in this work have been performed on this cluster.

In order to run an application on the LLSC, first the user launches a job. The job contains information about the computer resources needed for its execution. e.g. amount of memory, cpus, etc. Moreover, it includes details about the process itself such as: name, input and output.

In order to guarantee efficient and effective use of the resources, a scheduler is in charge of the management of the jobs in the cluster. The main goals of the scheduler are: (1) allocation of computer resources, (2) job execution and (3) report to the user the output of the execution.

To analyze large datasets, it has become typical to use clusters of machines to execute jobs consisting of many tasks. Jobs of many applications coexist on these clusters and their tasks have diverse resource demands (GAK+14).

The LLSC consists of:

- A single user node, or head node, running the scheduler, i.e. TORQUE (Sta06).
- 24 worker nodes with varying configurations:
  - 13 nodes with two dual-core Xeon 5150 CPUs and 16 GB main memory.
  - 9 nodes with two quad-core Xeon E5430 CPUs and 16 GB main memory.
  - 2 nodes with two dual-core Xeon 5150 CPUs and 8 GB main memory.
  - All nodes have 400 GB of local storage in a hardware RAID-0 configuration.
  - All nodes are interconnected with 100 MBit/s network interfaces.
- 2 file servers with 7.5 TB as temporary storage in hardware RAID-5 configuration, 32 GB main memory and connected to the network with a speed of 1 GBit/s.

The user node runs the TORQUE Resource Manager. All experiments are run through TORQUE as jobs. TORQUE is responsible for managing resources, the most important of which are the allocation of nodes and scheduling of jobs. TORQUE allows features such as requesting resources, tagging nodes, advanced job logging and statistics, job arrays and easy integration with third party parallel computing solutions such as MPI (Message Passing Interface).

5.3 Image Analysis

The experiments in this chapter utilize a standard dataset in cytomics and two Image Analysis algorithms: (1) Watershed Masked Clustering (YV12) and (2) Kernel Density Estimation (YVDvdW09), which are image segmentation and object tracking methods, respectively (CYW+11).

The images obtained from the image acquisition phase are digital images, and as such they can be processed using digital image processing techniques. The purpose of the experiments is to measure phenotypic properties. Since phenotype is defined as the observable characteristics of some object, having some way to determine the boundaries of that object is critical. The success of both algorithms is highly dependent upon this. One digital image processing technique designed for this purpose is image segmentation.

Segmentation is the process of separating an image into its constituent parts or objects. Usually this means separating the background from the foreground. In our image samples this is also the case. Each pixel belonging to a cell in the image is considered part of the foreground, and all other pixels are background. Segmentation

is considered one of the most difficult image processing tasks (GW06), and also one of the most important since a lot of subsequent processing techniques are dependent on the output of the segmentation phase. Separation of foreground and background is key to further decomposing the foreground into an accurate collection of object masks that represent individual cells.

It is important to note that the segmentation phase does not have to take the raw images directly from image acquisition. Segmentation is usually preceded by an Image enhancement phase. In this phase imperfections in the image such as too much noise or low contrast can be adjusted to make the image suitable for the segmentation tasks. Our algorithm makes use of enhancement techniques such as subtracting the background, gaussian blur, noise suppression and contrast enhancement. There are numerous popular segmentation techniques, and each comes with their own strengths and weaknesses. The choice of which technique to use largely depends on the composition of the target image.

Segmentation methods generally operate on one of two basic properties of intensity values: discontinuity and similarity. The former uses abrupt changes in an image to partition the image. The latter attempts to find regions of the image that are similar in pixel value to each other. Abrupt changes are usually edges of objects. For similar regions, the notion similar must first be defined. Put simply, for our input images obtained by fluorescence microscopy, similar pixels have close intensity values to neighbouring pixels above a certain threshold. The end result is a binary image. Each pixel was assigned intensity value 1 if it belongs to a cell, and 0 otherwise. The resulting image is referred to as *the mask*.

The segmentation step is followed by object tracking. Object tracking algorithms find links between objects over a time lapse series. This information can be used to study the movement of these objects over a period of time. In our case, a link must be found between two objects that appear in consecutive images. Both algorithms are described in detail in (Yan13).

5.4 Data Analysis and Pattern recognition

In (Ber03), data mining is defined as the process of identifying patterns and relationships in data that often are not obvious in large, complex data sets. As such, data mining involves pattern recognition and, by extension, pattern discovery.

Bergeron (Ber03) also identifies five major steps in the pattern recognition and discovery process: (1) *Feature selection*, (2) *Measurement*, (3) *Processing*, (4) *Feature extraction*, and (5) *Classification and discovery*.

Feature Selection. From the universe of available features, the selection of a set of features or attributes is considered the first step in pattern recognition.

Measurement. The measurement phase involves converting the original pattern into a representation that can be easily manipulated programmatically.

Processing. After the measurement process, the data are processed to remove noise and prepare for feature extraction. Processing typically involves executing a variety of error checking and correction routines, as well as specialized processes that depend on the nature of the data.

Feature Extraction. Feature extraction involves searching for global and local features in the data that are defined as relevant to pattern matching during feature selection. Clustering techniques, in which similar data are grouped together, often form the basis of feature extraction.

Classification and Discovery. In the classification phase, data are classified based on measurements of similarity with other patterns. These measurements of similarity are commonly based on either a statistical or a structural approach.

5.5 Implementation

Currently there are many open source software tools which assist researchers to perform complex image analysis processes in HTS experiments. For that reason it is relevant for systems that manage experiments data to facilitate the integratation of those tools to their architecture. CytomicsDB has been designed to assist the whole HTS workflow (YLL+11), thus the tools used during the analysis stage are easily integrated to our platform through web services. In this section, the architecture will be described as well as how the interaction between the components in the architecture are synchronized during the analysis stage.

5.5.1 The architecture

In Chapter 2 CytomicsDB's architecture was introduced, it has a four-layered style architecture: (1) *Presentation layer*, (2) *Service layer*, (3) *Persistence layer* and (4) *Repository*. This design allows scalability and flexibility thanks to the easy integration of external systems to the platform. In this particular case the integration of the cluster LLSC, built at Leiden University for assisting researchers in the execution of complex tasks during the image and data analysis where the large volume of data involved demands high computational resources. In this section the architecture is described from the image analysis point of view.

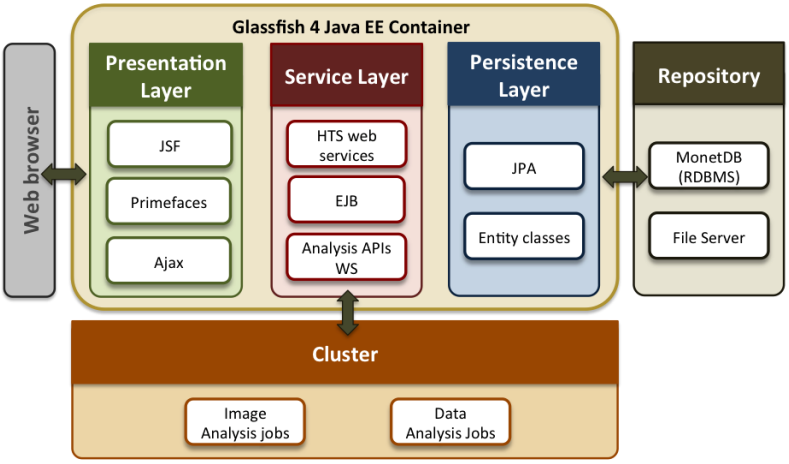


Figure 5.1: *CytomicsDB Architecture - Cluster Integration*

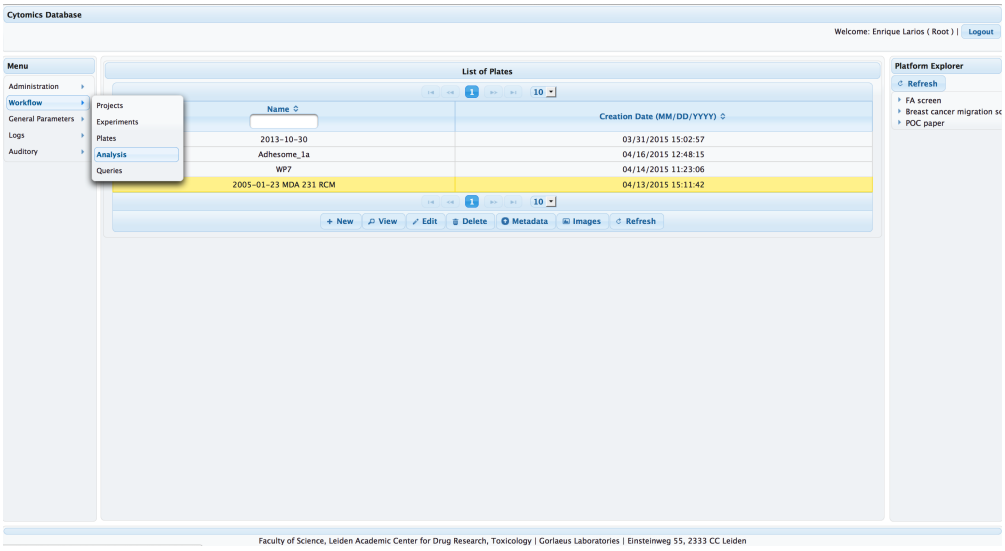


Figure 5.2: *Web Interface for Analysis*

The presentation layer

According to the HTS workflow, upon complexion of the image acquisition step and after uploading the images obtained to CytomicsDB, the analysis process can be triggered just by selecting a plate in the web interface and then selecting the option for analysis (Fig. 5.2). This layer also manages the web pages for visualizing the results, basically phenotype descriptors, binary masks and trajectories.

The service layer

This layer includes the web services in charge of the management of the analysis process. This style provides security and it is also a key factor in the integration with external systems. In this case the integration with the cluster LLSC is straightforward and independent of the programming language or the operating system running in each component of the architecture.

The persistence layer

The Java Persistence API (JPA) framework has been implemented in this layer to keep a bidirectional correspondence between the database and objects. Those Java objects used in this framework are known as Java Entities (KS06). The entities used in CytomicsDB map a physical table from the database and are the most secure way to manipulate and query the data stored in MonetDB.

The repository layer

The image analysis results consists basically of: (1) phenotype data, which is parsed to the relational database management system and linked to the original image dataset. (2) Binary masks and Trajectories, these results are also images that are stored in the File Server, and their location stored in MonetDB (Bon02) (LZCV14).

The Cluster

In Section 5.2 it was introduced the hardware used in the LLSC, the following describes how it works for the image analysis stage, in particular case for the segmentation and object tracking.

The input stacks are obtained from the Database. There are three main components involved. Two scripts written in Python named PA.py and PA-Seg.py, to setup the tracking and segmentation experiments respectively. Two TORQUE/PBS jobscripts using Bash called parallel-tracking.jobscript and parallel-segmentation.jobscript. Finally, a jar file named PA.jar containing the modified ImageJ source and Java implementations for segmentation and tracking. The ImageJ source is modified so that GUI components of certain plugins are no longer instantiated. The call hierarchy is PA.py → jobscript → PA.jar. PA stands for Phenotype Analysis. PA.py takes a text file as its argument. This text file itself contains arguments for the rest of the process. PA.py creates a global output directory for the job. Each node has read/write privileges for this directory. It copies a file containing the locations of all input stacks to this directory. It also copies the jobscript, depending on whether it is a tracking or segmentation job. The script also passes along any relevant arguments to the jobscript. The last statement in both Python scripts executes qsub, which is a TORQUE command and instructs TORQUE to schedule the jobscript. The jobscript submitted to the cluster contains

a header. In this header the output directory can be specified for all job output and logfiles. In our case this is the global output directory. These logfiles can later be used to gather job statistics. Job resources can also be requested. It is possible to request only nodes of a certain type or with a certain amount of memory. The main feature we are interested in is the amount of nodes, and the processors per node. Using these two attributes we can control the total number of resources available per job and measure how total job execution time reacts to different combinations of nodes and processors per node. The jobscript controls a selected number of nodes and is responsible for calling PA.jar on each node with the correct parameters. For segmentation, each node is given an equal number of stacks to process. If there is a remainder it is again split over the nodes. Once the node “knows" which stacks it must segment it can run concurrently with no synchronization until it is finished. For tracking, the jobscript first calculates the input arguments which it writes to a file on each node. Again, each node can process its stacks concurrently. When all nodes are finished the joining operation is started. PA.jar takes different arguments for segmentation and tracking. For segmentation it takes the input stack, the number of cores to use and the location of the output directory. Tracking takes the input stack, the masked input stack, the output directory, the number of cores, and the id’s of the slices it must process.

MonetDB.R Package

R programming language has become for both academia and industry one of the most important tools for data analytics, statistics, visualization and data science. Scientist use R to solve their problems for data processing specially with respect to large volumes of scientific data. R is also becoming important because it is not only very flexible in reading, manipulating, and writing data but all its outcomes are directly available as objects for further programming (Kri09).

One of the challenges of data management in Cytomics mentioned in Chapter 1 is *the movement of data*, and the need for processing the data “in place" and transmit only the resulting information. Following this paradigm, MonetDB.R package was developed for providing a transparent connection to MonetDB from the R software suite. MonetDB.R was designed to reduce the overhead of shuffling data between different systems, resulting in the improvment of data processing time. This is achieved because the package decides which portions of the data analysis should be performed by either R or the MonetDB database. Thus, the combination of both applications in CytomicsDB platform speed up the process of data discovery (Mon).

5.5.2 The image and data analysis data flow

The diagram shown in Figure 5.3 illustrates the flows of the control in the HTS image and data analysis stage. In case of the image analysis, Figure 5.3 shows the steps for the execution as typical cases of the segmentation and object tracking algorithms. On

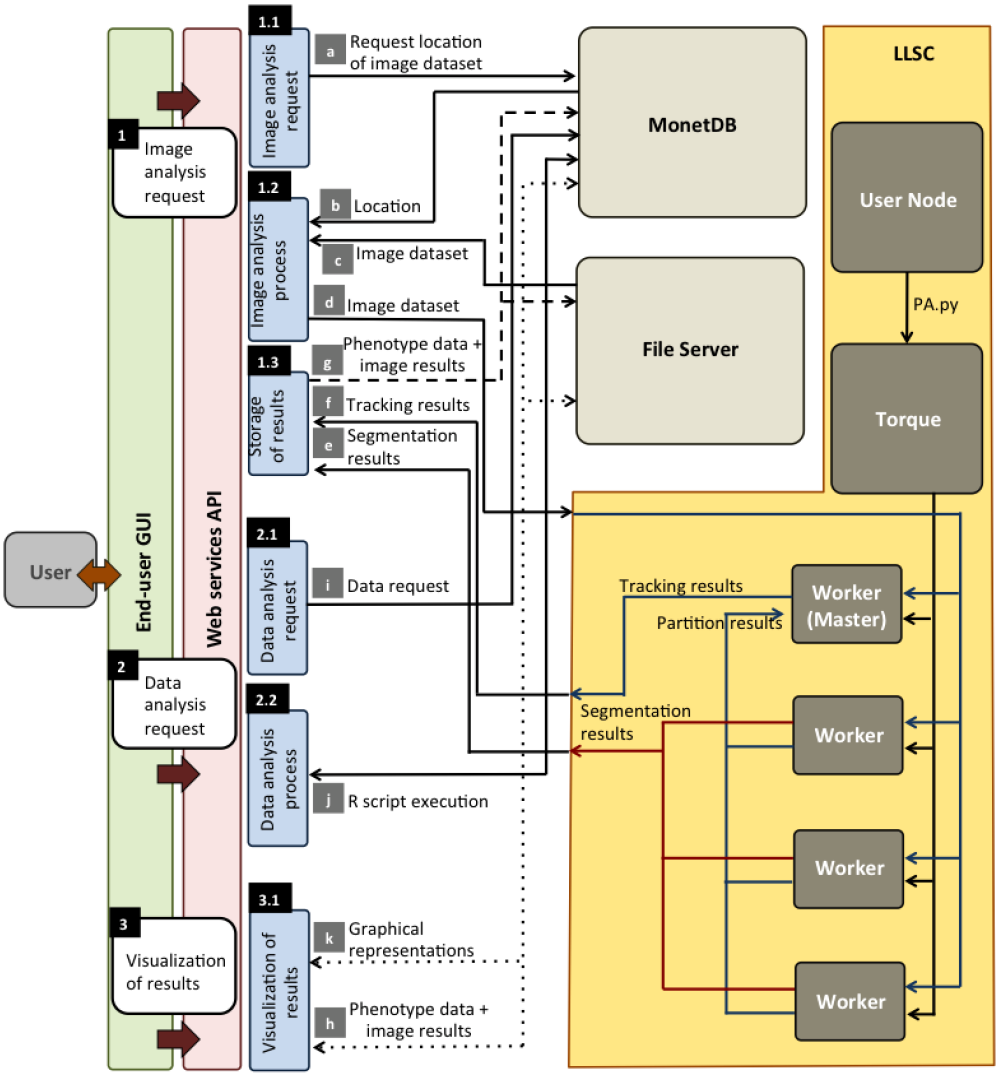


Figure 5.3: *Image and Data analysis data flow*

the other hand, in case of the data analysis, it is shown the steps for the execution of the R scripts in order to obtain graphical representations of the large datasets and identify patterns. The main three features of this stage are (1) *the image analysis request*, (2) *the data analysis request* and (3) *the visualization of the results*. How this main features are executed is shown by three sequences of annotated arrows starting from the end-user GUI. Arrows handling the same operation are grouped together by a major number, while the alphabetical characters correspond to the order of a particular step that is

called in its containing sequence.

In Figure 5.3, we describe each of the sequences. Sequence 1 handles the image analysis request, this request is made from the web interface in CytomicsDB. First, the request is sent to MonetDB, the location of the image datasets under analysis is sent to the web service in charge of the management of the analysis process. Second, the image dataset is extracted from the File Server and then sent to the LLSC for the execution of the segmentation and then object tracking tasks. Finally, upon the complexion of the analysis process the phenotye data is stored in MonetDB and the new image datasets resulted are stored on the File Server, then their location is stored in MonetDB for further querying. Sequence 2 handles a data analysis request, which is first sent to MonetDB. Subsequently, the data analysis process is triggered. R scripts are executed using the package MonetDB.R, this package will manage which part of the data analysis will be performed in R or in MonetDB database. Sequence 3 handles the visualization of the results, since the results and experiment's metadata is stored in one single place, the visualization of the results can be handled by just requesting the data of the analysis results from MonetDB. In the web interface, the results are displayed with the corresponding plate layout.

5.5.3 Image Analysis setup and Results

Segmentation

This experiment measures job time for an input set of 512 stacks, processed over 1,2,4 or 8 nodes. The images are MTLn3 cancer cells, cultured in 4 different well plates. The 512 stacks are made up of:

- 144 images from well plate 2, not treated.
- 144 images from well plate 2, treated with EGF.
- 144 images from well plate 3, not treated.
- 88 images from well plate 3, treated with EGF.

This experiment utilizes all available nodes and was run 3 times. The results reflect the average of these runs. There are also an equal amount of jobs as nodes. Figure 5.4 shows how segmentation scales for up to 24 nodes, confirming a linear speedup.

Object tracking

The experiment input is 144 non treated image stacks from well plate 2, and their corresponding masked versions obtained through prior segmentation. Each stack has 31 slices. This experiment processes partitions of a stack concurrently but does not process multiple stacks concurrently. Each image file is read directly from the File Server and tracked. Once all stacks have been tracked and have written their

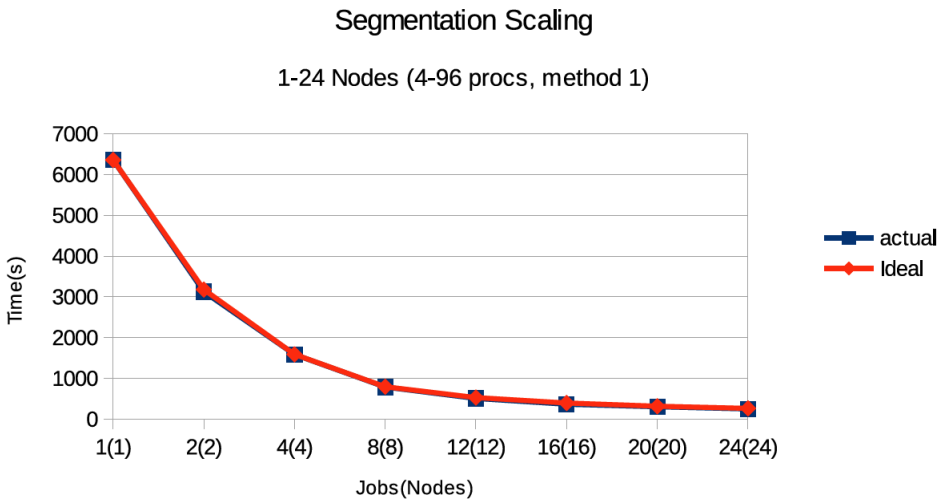


Figure 5.4: *Segmentation results using up to 24 nodes*

local results the joining starts. Each node sends its results to a master node where the joining algorithm is run for all stacks. When all algorithms are finished all results are copied back to the File Server. The tracking, joining and copy operations are all timed separately. The combinations listed in Table 5.1 were tested.

Table 5.1: *Algorithms tested*

Algorithm	Cores	Nodes
Sequential tracking	1	1
Partitioned tracking	1	2-10
Partitioned tracking	2	1-6
Partitioned tracking	4	1-3

Each experiment is run 3 times and the averaged results are given in figure 5.5. The line represents a linear (ideal) speedup. The chart shows that there is in fact a significant improvement even when more than 2 processors are used. In some cases a superlinear speedup is achieved. The use of 1 core per node consistently gives the best performance. A possible explanation for this is that 1 cpu is used for the algorithm, and in our implementation the entire node (4 processors) has been reserved by TORQUE. No other user intensive user processes were running on the remaining 3 cores. As a result, when 1 core per node is used the entire cpu cache and RAM memory can be used for tracking without interference. When 2 or 4 cores per node are used these resources must be shared, possibly causing increased cache miss rates. Using 4 cores

per node is the slowest option (though it still achieves close to linear speedup). Even though it is outperformed, using 4 cores per node is still desirable. If one core per node is used and another user requests the remaining cores for another process performance may drop beyond that of using 4 cores per node. Even if no such request is made the remaining cores are essentially wasted. It is worth noting that using 7, 8 or 9 processors seems to be worse than using 6. This can be explained by the inefficiency of the slice allocator. Each node is allocated:

$$\frac{sizeofstack}{nrofnodes} + 1$$

slices, except the last which also processes the remainder. For 6 nodes, each node processes $(31/6) + 1 = 6$ slices. For 7 nodes, each node processes $(31/7) + 1 = 5$ slices, except the last which processes 8 slices. Similarly, 8 nodes each process 4 slices, except the last which processes 11. Each node must complete before the next stack can be processed. Therefore, when using 7,8 and 9 nodes since there is a node that processes more slices than when using 6 nodes, 6 nodes is faster. This method of slice allocation is the most basic easiest to implement method and should be improved in a future version.

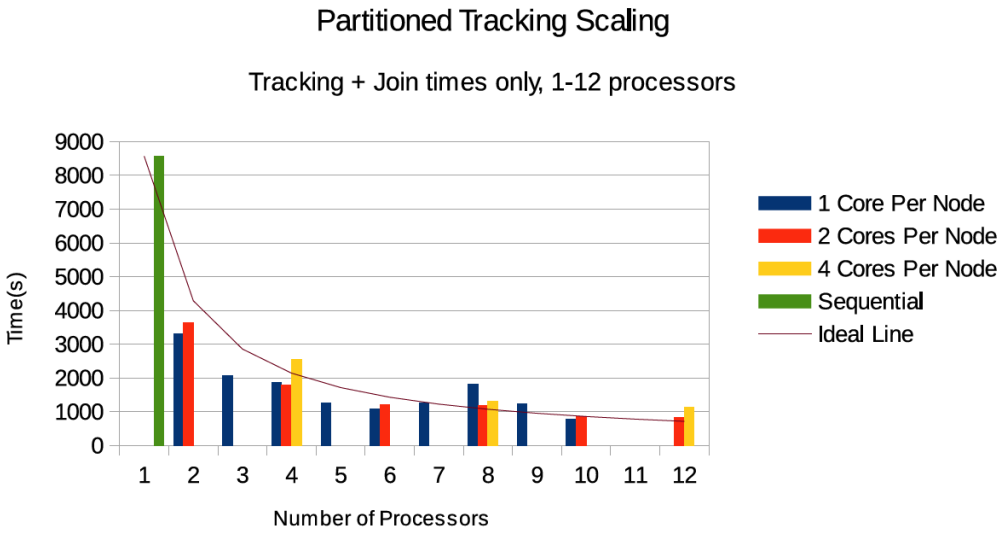


Figure 5.5: *Partitioned concurrent tracking and join times*

5.5.4 Data Analysis setup

In Section 3.3, it was described a case study which explains how the data model designed in CytomicsDB supports the image analysis stage. Tables *Feature* and *Measurement* are used to store the phenotype data resulted from the image analysis stage, and

become the base for starting the data analysis process. In Figure 2.11, it was shown the the database schema for storing the measurements metadata. This section is based on the case study introduced in Section 3.3. The aim of the case study is to investigate the process of endocytosis and epidermial growth factor receptor (EGFR) signaling. EGFR signaling triggers breast cancer cells to escape from the primary tumor and spread to the lung, resulting in poor disease diagnosis. Moreover, it may result in resistance to anti-cancer therapy (CYW$^+$11).

Based on the data stored in the tables *Feature* and *Measurement*, the data analysis is triggered. First, it is requested from the database the feature names and the feature values used in this experiment.

The query executed for retrieving the feature names is:

```
SELECT f.feat_name
FROM HTS.Feature_plate p, HTS.Feature f
WHERE p.feat_id = f.feat_id and  p.plat_id = 17;
```

The *Features* are associated to a plate, and in this experiment is being used the plate identified by *plate_id* with value "17". The result set obtained is:

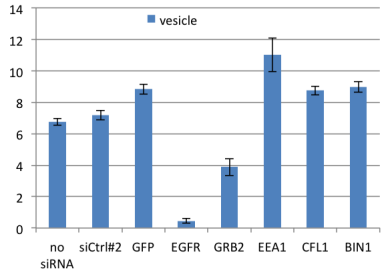
```
testNr, frame#, obj#, area, perimeter, massCenterX, massCenterY, extension, dispersion,
elongation, orientation, compactFactor, averageIntensity, Nucleolus Dist, Nuke X, Nuke Y,
Number of FAK, Number of Nucleolus, In Nucleolus, Closest FA Dist, long axis,short axis,
Border Distance, Int Std, Int Smoothness, Int Skewness, Int Uniformity, Int Entropy
```

Thanks to the use of the MonetDB.R package, there is no need to retrieve the feature values a.k.a. measurements from MonetDB to the web service layer in order to perform the data analysis. For this particular experiment, the table *measurement* contains 279 036 rows and 28 features.

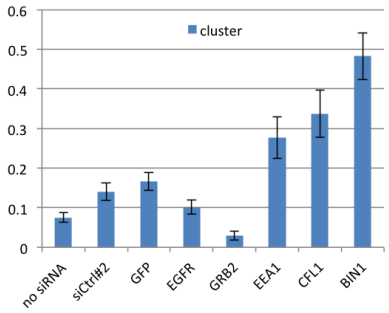
A view of the features value stored in the table *Measurement* can be retrieved executing the following query:

```
SELECT f.feat_name, m.meas_value
FROM HTS.Feature_plate p, HTS.Feature f, HTS.Measurement m
WHERE p.feat_id = f.feat_id and p.feat_id = m.feat_id and
      p.plat_id = m.plat_id and p.plat_id = 17;
```

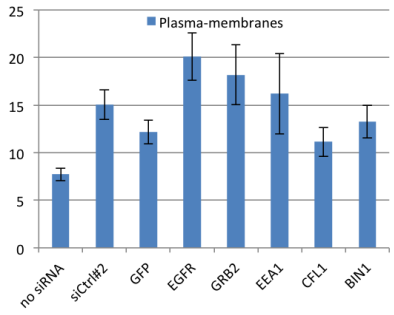
The data analysis process begins with the execution of the R script. This script is in charge of the following steps: (1) *Classification* and (2) *Comparison of the treatments per well*. In the *Classification*, for this particular case study (Section 3.3), the measurements were classified in three subsets: Cluster, junction and vesicle. In order to have this classification completed, first a ground truth data preparation is elaborated, this will be used for training the classifier algorithm, then it is necessary to perform feature selection and feature extraction and finally the feature values will be classified in one of these subsets. Upon completion of the classification task it is possible to do the comparison of the treatments per well and identify: (1) *Number of vesicles per nucleus* (c.f. Figure 5.6(a)), (2) *Number of clusters per nucleus* (c.f. Figure 5.6(b)), and (3) *Plasma-membranes (pixel) per nucleus* (c.f. Figure 5.6(c)) (CYW$^+$11).



(a) Number of vesicles per nucleus



(b) Number of clusters per nucleus



(c) Plasma-membranes (pixel) per nucleus

Figure 5.6: *Comparison of results with three phenotypic groups*

5.6 Conclusions and Future work

The use of cluster computing in Bioinformatics research, especially in cytomics, has been proved to be highly necessary due to the large datasets generated in HTS workflows. Due to the advantage of using parallelism in cluster architectures, traditional algorithms used for instance in image analysis need to be adapted to such environments. In this chapter, we have presented the cluster integration to CytomicsDB architecture and explored the effect of parallel computing on the performance of these algorithms in HTS experiments. We have shown that both segmentation and tracking algorithms can be parallelized efficiently with segmentation scaling linearly up to at least 96 processors using a combination of stack and slice level concurrency. Additionally, it has been described for the case study presented in Section 3.3 how the steps for data analysis are accomplished. Finally, it is still possible to reduce the volume of data generated in the image analysis stage e.g. auxiliary images such as binary masks and trajectories. Instead of storing the image files as tiff files, a matrix can be used to store the location of the objects of interest.