SEARCH STRING PROJECT REPORT

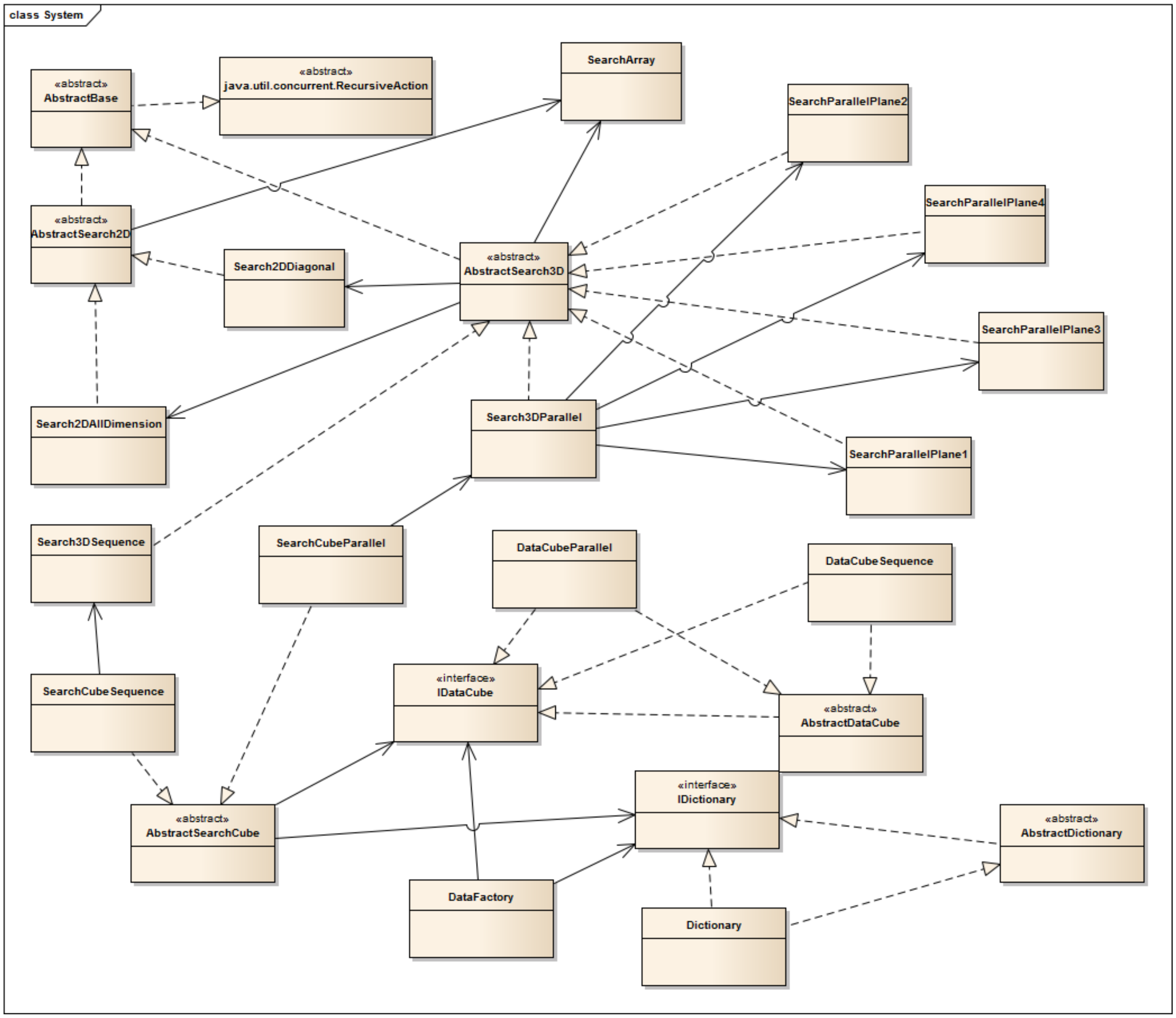
Contents

1. **Approach method**
    - I go straight to implement the parallel search at first instead sequence because I am not sure if ForkJoin can give me an expected result. After finishing implementing APIs that shared between Parallel search and Sequence search, I decided going with Parallel in case ForkJoin is impossible to successfully implement for the project, I can make a change soon.
    - After finishing Parallel part, I notices that the difference between Parallel and Sequence is the way to execute a task. So in Parallel search, all scheduled tasks are executed at the same time meanwhile in sequence search, the next task will be executed after the first task is completed and the given string have not found yet. I think in this way, my project can reuse much code.
2. **Search Algorithm**
    - I decided to divide 3D cube into planes of 2D array as following:
        - Selects all planes that are perpendicular to Z dimensions. Searching the given string in all 2D arrays with 3 dimensions: horizontal, vertical and diagonal of all 2D arrays.
        - Selects all planes that are perpendicular to Y dimensions: Searching the given string in all 2D arrays with only 1 diagonal dimension because of avoiding the duplicated points with planes that are perpendicular to Z dimensions.
        - Selects all planes that are perpendicular to X dimensions: Searching the given string in all 2D arrays with only 1 diagonal dimension because of avoiding the duplicated points with planes that are perpendicular to Z dimensions.
        - Selects all planes that have coordinators : x = x0 + 1, y = y0 - 1, z = z0 + 1 and compares the the given string with letters laying these planes.
        - Selects all planes that have coordinators : x = x0 + 1, y = y0 + 1, z = z0 + 1 and compares the the given string with letters laying these planes.
        - Selects all planes that have coordinators : x = x0 + 1, y = y0 - 1, z = z0 - 1 and compares the the given string with letters laying these planes.
        - Selects all planes that have coordinators : x = x0 + 1, y = y0 + 1, z = z0 - 1 and compares the the given string with letters laying these planes.

- Search a string in 2D array is smallest unit in the design and can be considered as a scheduled task when program is executed.
- I use ForkJoin framework to arrange the these tasks and executed them in parallel or in sequence.

3. **Why did I choose ForkJoin framework to implement parallel computing**
    - I prefer Java to the others
    - Framework design can solve my problem.
    - MapReduce algorithm is used to develop ForkJoin is well known.
    - There are some open source parallel computing for Java researched by some organizations such as ParallelJ, Pyjama, MPJ express but they are still potential and been experiencing
    - APIs Javadocs for ForkJoin are much understandable and usable.

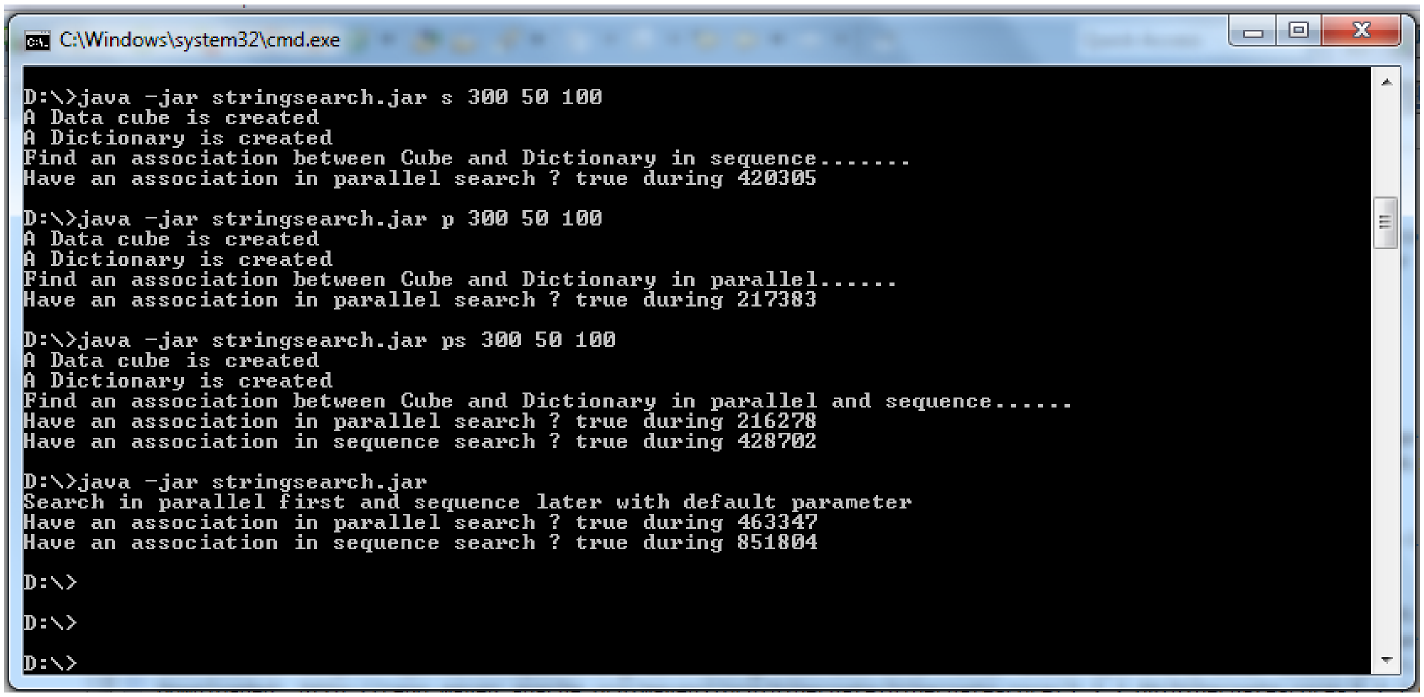
4. **Architecture Overview**



5. Testing User

- User can enter input parameter manually as the size of cube, the size of dictionary, the length of word and choose the type of search (in parallel, in sequence or both)
    - The first argument:
        - The type of search:
        - Value: p (parallel) or s (sequence) or ps (both) or sp (both)
    - The second argument:
        - The size of cube:
        - Value is an integer between 3 and 1000
    - The third argument:
        - The size of dictionary
        - Value: is an integer between 3 and 1000

        - The fourth argument:
            - The length of each word in the dictionary
            - Value is an integer between 2 and 100
    - If user enters nothing, program will run with generating randomly parameters:
        - Search the dictionary in the cube with parallel and sequence.
        - Cube size is generated randomly between 3 and 1000.
        - Dictionary size is generated randomly between 3 and 1000.
        - Word length is is generated randomly between 2 and 100.
    - Command :
        - Go to src/jar/ folder
        - Enter : java -jar stringsearch.jar [type] [cube size] [dictionary size] [word length]
    - There are some images below as examples:



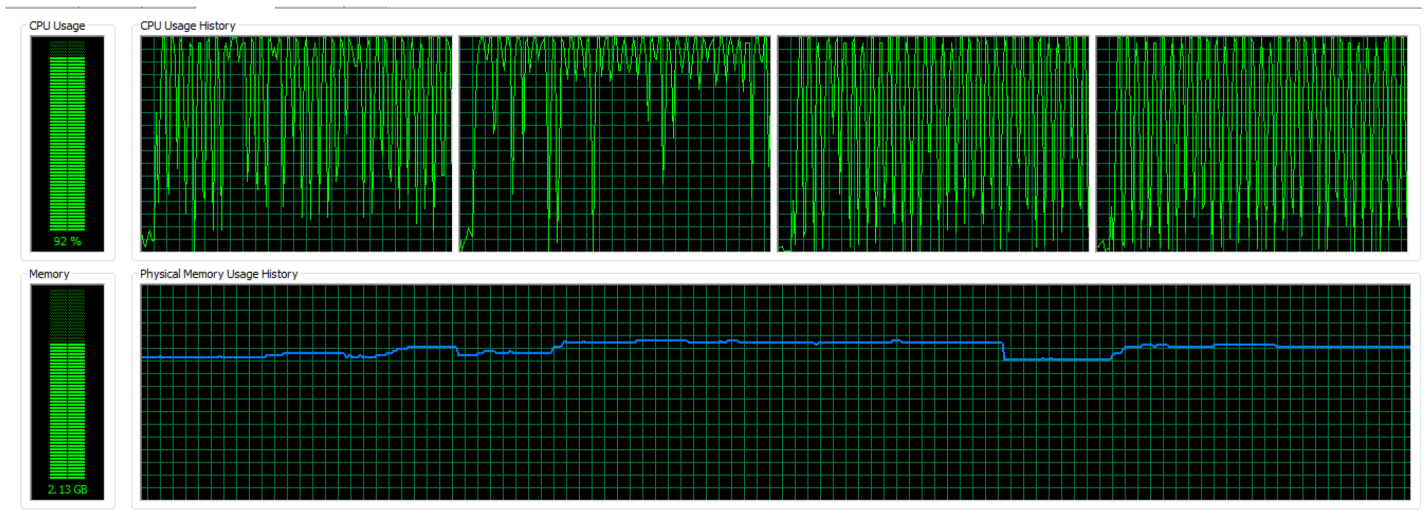
```
D:\>java -jar stringsearch.jar s 300 50 100
A Data cube is created
A Dictionary is created
Find an association between Cube and Dictionary in sequence.......
Have an association in parallel search ? true during 420305

D:\>java -jar stringsearch.jar p 300 50 100
A Data cube is created
A Dictionary is created
Find an association between Cube and Dictionary in parallel......
Have an association in parallel search ? true during 217383

D:\>java -jar stringsearch.jar ps 300 50 100
A Data cube is created
A Dictionary is created
Find an association between Cube and Dictionary in parallel and sequence......
Have an association in parallel search ? true during 216278
Have an association in sequence search ? true during 428702

D:\>java -jar stringsearch.jar
Search in parallel first and sequence later with default parameter
Have an association in parallel search ? true during 463347
Have an association in sequence search ? true during 851804

D:\>

D:\>

D:\>
```

6. **Performance comparison**
    - **Environment: Windows 7, core 4, 4GB ram (2.93 usable) and there are some applications and services running when taking these tests.**
        - **Speed**

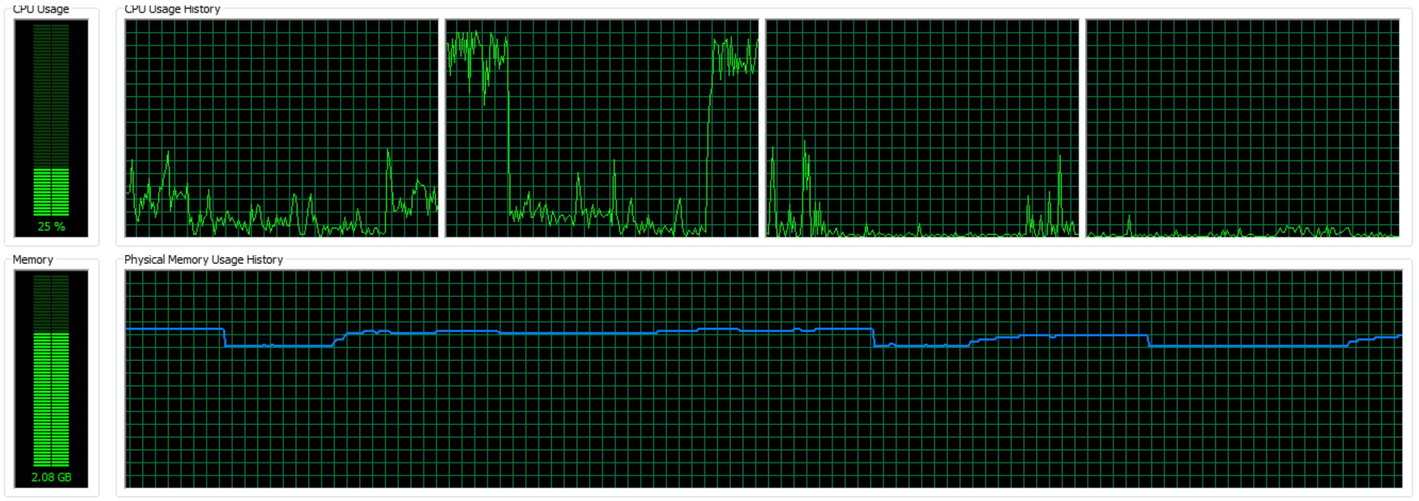
Parameter	Parallel (milliseconds)	Sequence (milliseconds)	Speedup
Cube size: 300 Dictionary size: 100	432321	841305	1.94601927734253

Word length: 10			
Cube size: 400 Dictionary size: 200 Word length: 8	2252906	4079882	1.810941956743868
Cube size: 200 Dictionary size: 100 Word length: 100	23771	46260	1.946068739220058

- **CPU usage**
  - **Parallel**



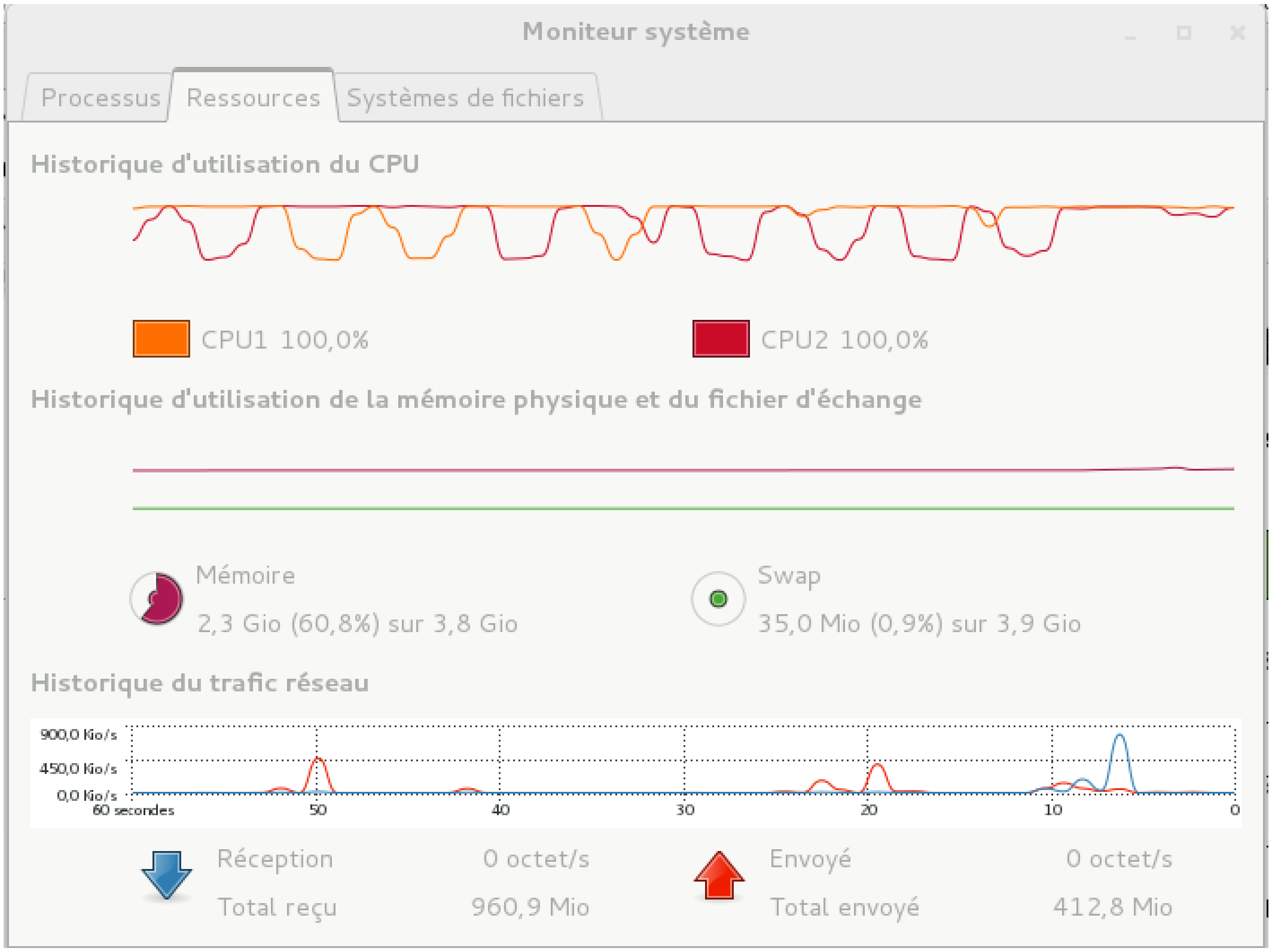
  - **Sequence**



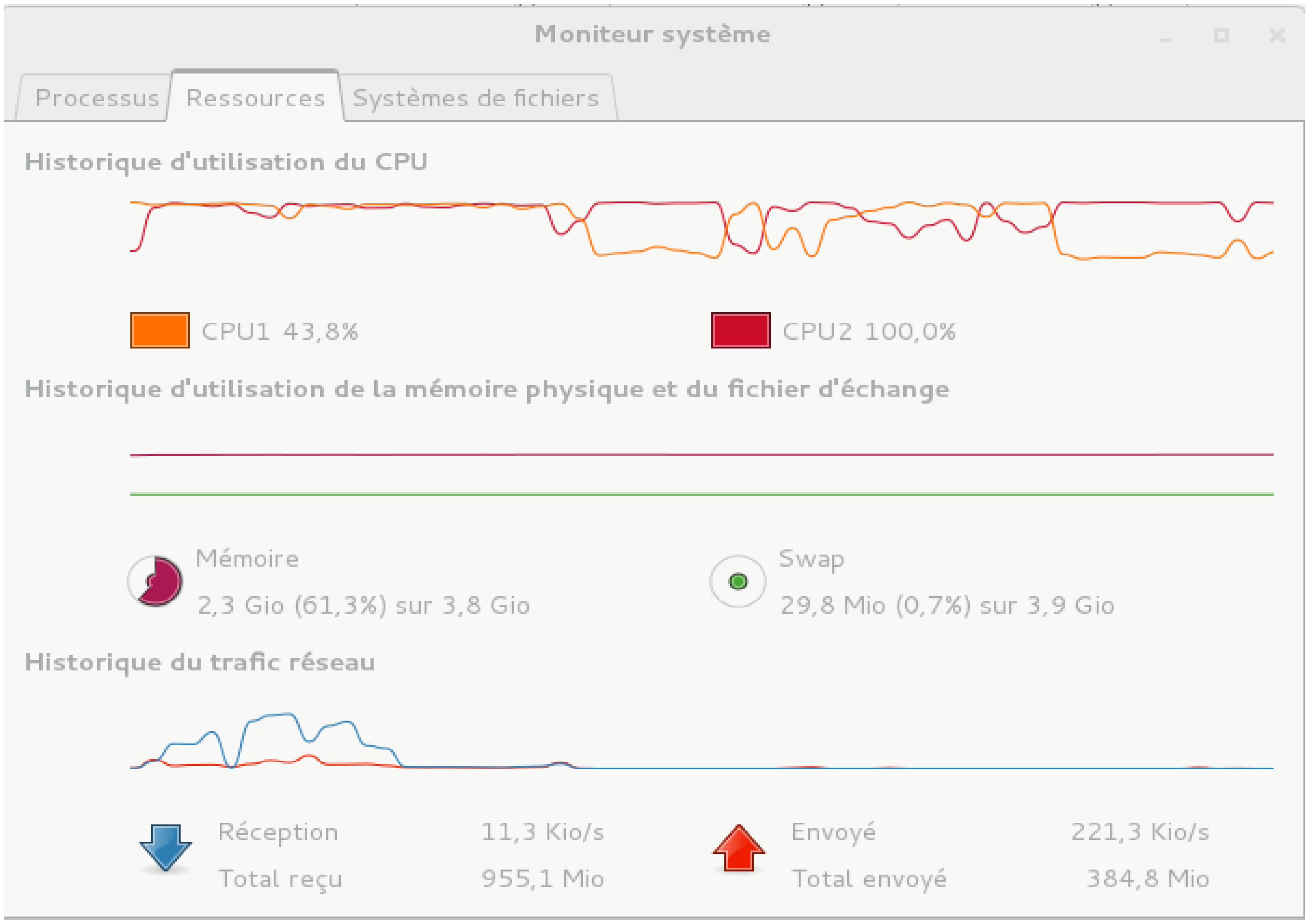
- **Linux: Pentium(R) Dual - Core, RAM 4GB**
  - **Speed: It seems that search string application runs faster and consumes less memory in Linux than Windows 7.**

Parameter	Parallel (milliseconds)	Sequence (milliseconds)	Speedup
Cube size: 300 Dictionary size: 100 Word length: 10	230593	356323	1,545246386
Cube size: 400 Dictionary size: 200 Word length: 8	1174760	1415859	1,205232558
Cube size: 200 Dictionary size: 100 Word length: 100	68093	99795	1,465569148

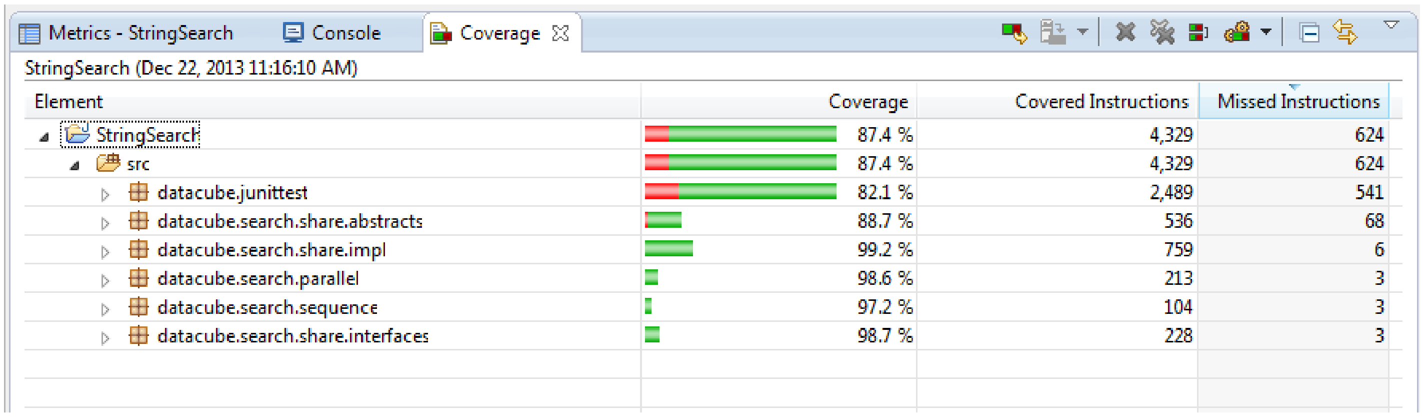
- **CPU usage**
  - **Parallel**



- **Sequence**


Moniteur système
Processus
Ressources
Systèmes de fichiers
Historique d'utilisation du CPU
CPU1 43,8%
CPU2 100,0%
Historique d'utilisation de la mémoire physique et du fichier d'échange
Mémoire
2,3 Gio (61,3%) sur 3,8 Gio
Swap
29,8 Mio (0,7%) sur 3,9 Gio
Historique du trafic réseau
Réception
11,3 Kio/s
Total reçu
955,1 Mio
Envoyé
221,3 Kio/s
Total envoyé
384,8 Mio

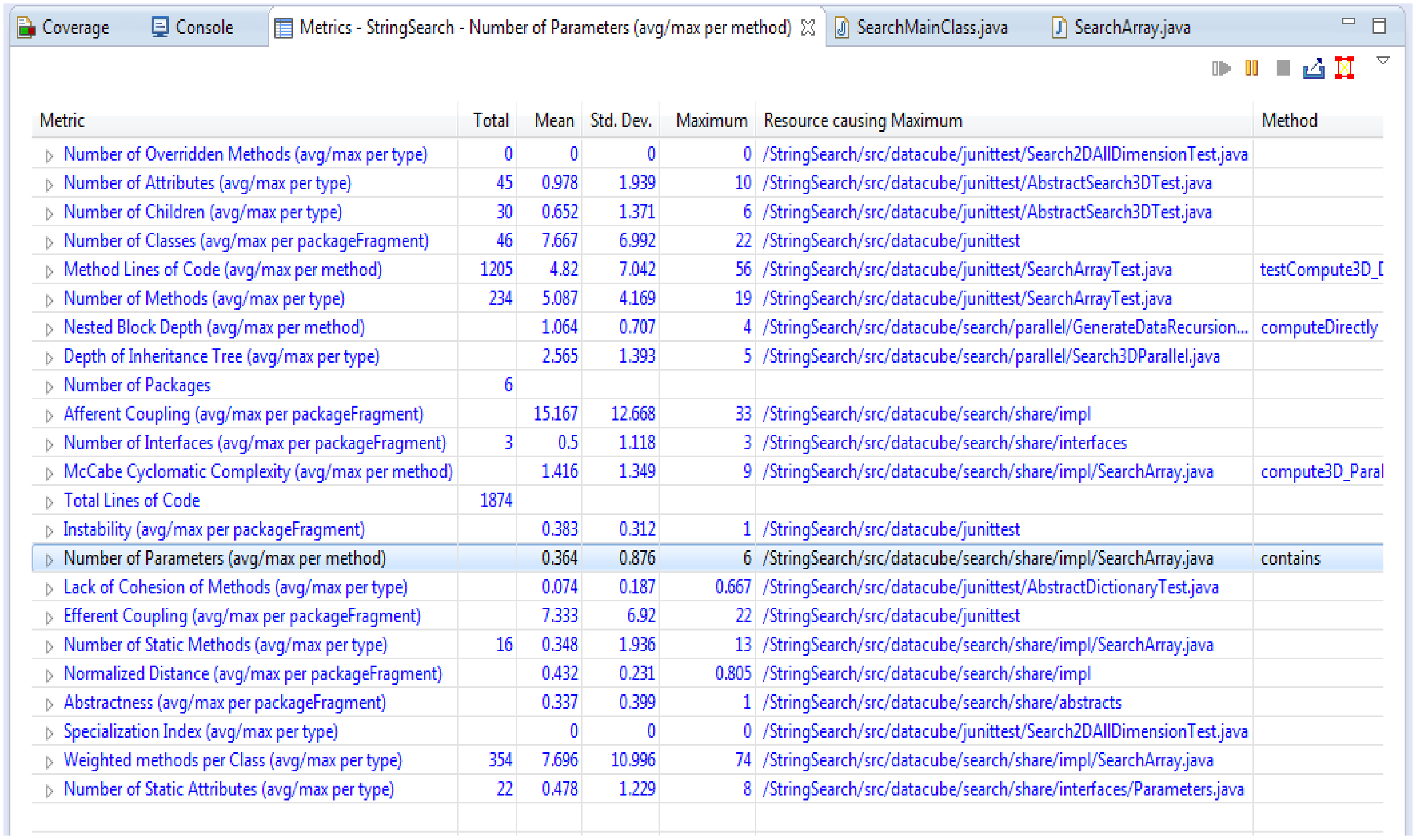
7. **Project quality report**
   - **Coverage :**



StringSearch (Dec 22, 2013 11:16:10 AM)

Element	Coverage	Covered Instructions	Missed Instructions
StringSearch	87.4 %	4,329	624
src	87.4 %	4,329	624
datacube.junittest	82.1 %	2,489	541
datacube.search.share.abstracts	88.7 %	536	68
datacube.search.share.impl	99.2 %	759	6
datacube.search.parallel	98.6 %	213	3
datacube.search.sequence	97.2 %	104	3
datacube.search.share.interfaces	98.7 %	228	3

   - **Metrics:**


Coverage
Console
Metrics - StringSearch - Number of Parameters (avg/max per method)
SearchMainClass.java
SearchArray.java

Metric	Total	Mean	Std. Dev.	Maximum	Resource causing Maximum	Method
Number of Overridden Methods (avg/max per type)	0	0	0	0	/StringSearch/src/datacube/junittest/Search2DAllDimensionTest.java	
Number of Attributes (avg/max per type)	45	0.978	1.939	10	/StringSearch/src/datacube/junittest/AbstractSearch3DTest.java	
Number of Children (avg/max per type)	30	0.652	1.371	6	/StringSearch/src/datacube/junittest/AbstractSearch3DTest.java	
Number of Classes (avg/max per packageFragment)	46	7.667	6.992	22	/StringSearch/src/datacube/junittest	
Method Lines of Code (avg/max per method)	1205	4.82	7.042	56	/StringSearch/src/datacube/junittest/SearchArrayTest.java	testCompute3D_[
Number of Methods (avg/max per type)	234	5.087	4.169	19	/StringSearch/src/datacube/junittest/SearchArrayTest.java	
Nested Block Depth (avg/max per method)		1.064	0.707	4	/StringSearch/src/datacube/search/parallel/GenerateDataRecursion...	computeDirectly
Depth of Inheritance Tree (avg/max per type)		2.565	1.393	5	/StringSearch/src/datacube/search/parallel/Search3DParallel.java	
Number of Packages	6					
Afferent Coupling (avg/max per packageFragment)		15.167	12.668	33	/StringSearch/src/datacube/search/share/impl	
Number of Interfaces (avg/max per packageFragment)	3	0.5	1.118	3	/StringSearch/src/datacube/search/share/interfaces	
McCabe Cyclomatic Complexity (avg/max per method)		1.416	1.349	9	/StringSearch/src/datacube/search/share/impl/SearchArray.java	compute3D_Paral
Total Lines of Code	1874					
Instability (avg/max per packageFragment)		0.383	0.312	1	/StringSearch/src/datacube/junittest	
Number of Parameters (avg/max per method)		0.364	0.876	6	/StringSearch/src/datacube/search/share/impl/SearchArray.java	contains
Lack of Cohesion of Methods (avg/max per type)		0.074	0.187	0.667	/StringSearch/src/datacube/junittest/AbstractDictionaryTest.java	
Efferent Coupling (avg/max per packageFragment)		7.333	6.92	22	/StringSearch/src/datacube/junittest	
Number of Static Methods (avg/max per type)	16	0.348	1.936	13	/StringSearch/src/datacube/search/share/impl/SearchArray.java	
Normalized Distance (avg/max per packageFragment)		0.432	0.231	0.805	/StringSearch/src/datacube/search/share/impl	
Abstractness (avg/max per packageFragment)		0.337	0.399	1	/StringSearch/src/datacube/search/share/abstracts	
Specialization Index (avg/max per type)		0	0	0	/StringSearch/src/datacube/junittest/Search2DAllDimensionTest.java	
Weighted methods per Class (avg/max per type)	354	7.696	10.996	74	/StringSearch/src/datacube/search/share/impl/SearchArray.java	
Number of Static Attributes (avg/max per type)	22	0.478	1.229	8	/StringSearch/src/datacube/search/share/interfaces/Parameters.java	

8. Hardware requirement

To run with maximum input parameter, I propose a computer with approximately 4GB usable free space