Using MapReduce for Large-scale Medical Image Analysis

HISB 2012

Presented by : Roger Schaer - HES-SO Valais

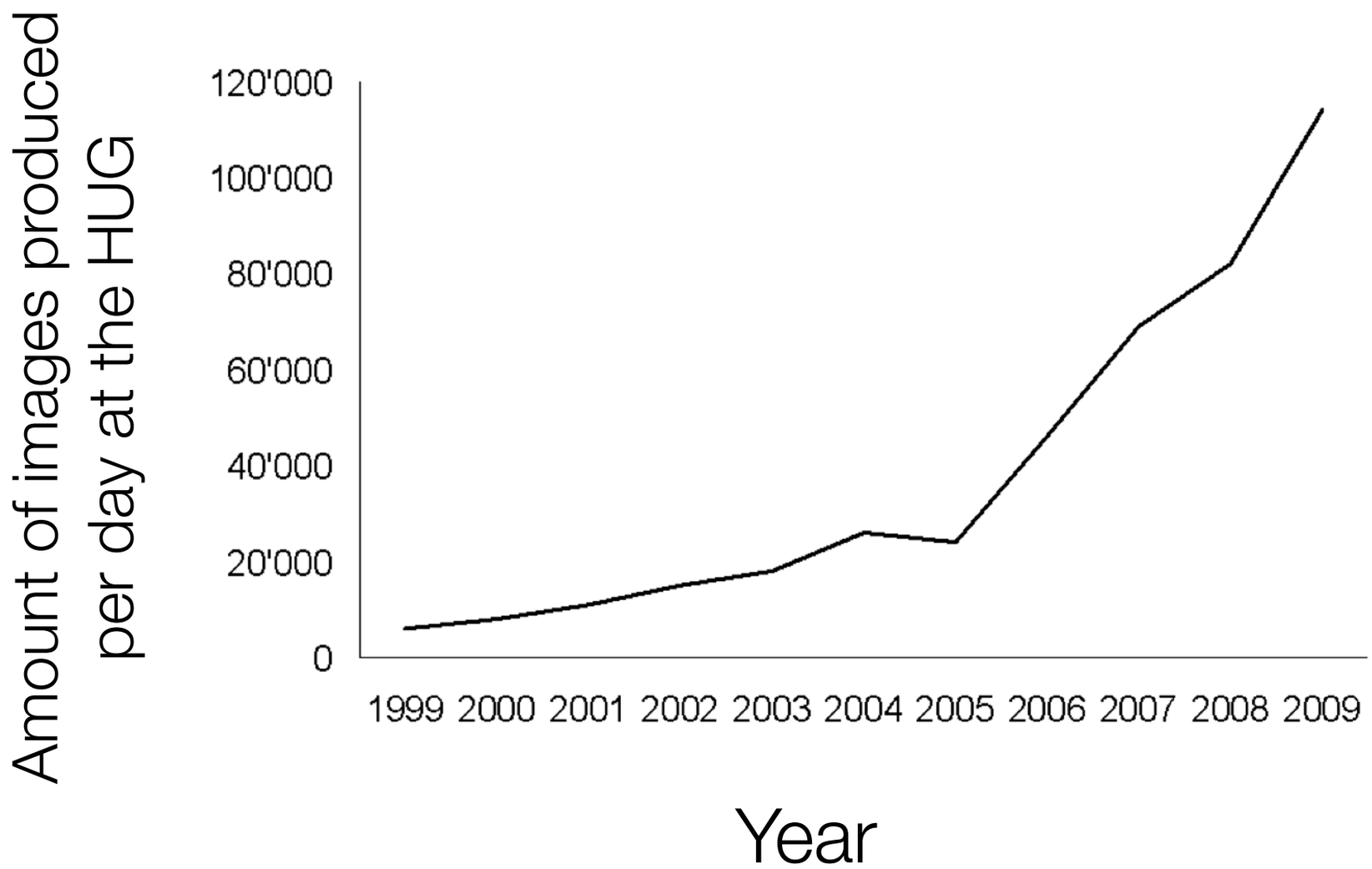
Summary

- Introduction
- Methods
- Results & Interpretation
- Conclusions

Introduction

Introduction

- **Exponential growth** of imaging data (past 20 years)



Introduction (continued)

- Mainly caused by :
  - Modern imaging techniques (3D, 4D) : **Large files** !
  - **Large collections** (available on the Internet)
- Increasingly **complex algorithms** make processing this data more challenging
- Requires a lot of **computation** power, **storage** and network **bandwidth**

Introduction (continued)

- Flexible and scalable infrastructures are needed
- Several approaches exist :
  - **Single, powerful machine**
  - **Local cluster / grid**
  - Alternative infrastructures (Graphics cards)
  - Cloud computing solutions
- First two approaches have been tested and compared

Introduction (continued)

- 3 large-scale medical image processing use cases
  - Parameter optimization for Support Vector Machines
  - Content-based image feature extraction & indexing
  - 3D texture feature extraction using the Riesz transform
- **NOTE : I mostly handled the infrastructure aspects !**

Methods

Methods

- **MapReduce**
- **Hadoop Cluster**
- Support Vector Machines
- Image Indexing
- Solid 3D Texture Analysis Using the Riesz Transform

MapReduce

- MapReduce is a **programming model**
  - Developed by Google
  - Map Phase : Key/Value pair input, Intermediate output
  - Reduce phase : For each intermediate key, process the list of associated values
- Trivial example : Word Count application

MapReduce : WordCount

MapReduce : WordCount

Input

MapReduce : WordCount

Input

#1 hello world
#2 goodbye world
#3 hello hadoop
#4 bye hadoop
...

MapReduce : WordCount

INPUT

#1 hello world
#2 goodbye world
#3 hello hadoop
#4 bye hadoop
...

MAP

MapReduce : WordCount

INPUT

#1 hello world
#2 goodbye world
#3 hello hadoop
#4 bye hadoop
...

MAP

MapReduce : WordCount

INPUT

#1 hello world
#2 goodbye world
#3 hello hadoop
#4 bye hadoop
...

MAP

hello 1

MapReduce : WordCount

Input	**Map**
#1 hello world	hello 1
#2 goodbye world	world 1
#3 hello hadoop	
#4 bye hadoop	
...	

MapReduce : WordCount

Input	**Map**
#1 hello world	hello 1
#2 goodbye world	world 1
#3 hello hadoop	
#4 bye hadoop	
...	

MapReduce : WordCount

Input	**Map**
#1 hello world	hello 1
#2 goodbye world	world 1
#3 hello hadoop	goodbye 1
#4 bye hadoop	
...	

MapReduce : WordCount

Input	**Map**
#1 hello world	hello 1
#2 goodbye world	world 1
#3 hello hadoop	goodbye 1
#4 bye hadoop	world 1
...	

MapReduce : WordCount

INPUT	MAP
#1 hello world	hello 1
#2 goodbye world	world 1
#3 hello hadoop	goodbye 1
#4 bye hadoop	world 1
...	

MapReduce : WordCount

Input	**Map**
#1 hello world	hello 1
#2 goodbye world	world 1
#3 hello hadoop	goodbye 1
#4 bye hadoop	world 1
...	hello 1

MapReduce : WordCount

Input	**Map**
#1 hello world	hello 1
#2 goodbye world	world 1
#3 hello hadoop	goodbye 1
#4 bye hadoop	world 1
...	hello 1
	hadoop 1

MapReduce : WordCount

Input	**Map**
#1 hello world	hello 1
#2 goodbye world	world 1
#3 hello hadoop	goodbye 1
#4 bye hadoop	world 1
...	hello 1
	hadoop 1

MapReduce : WordCount

Input	**Map**
#1 hello world	hello 1
#2 goodbye world	world 1
#3 hello hadoop	goodbye 1
#4 bye hadoop	world 1
...	hello 1
	hadoop 1
	bye 1

MapReduce : WordCount

Input	**Map**
#1 hello world	hello 1
#2 goodbye world	world 1
#3 hello hadoop	goodbye 1
#4 bye hadoop	world 1
...	hello 1
	hadoop 1
	bye 1
	hadoop 1

MapReduce : WordCount

INPUT	MAP	REDUCE
#1 hello world	hello 1	
#2 goodbye world	world 1	
#3 hello hadoop	goodbye 1	
#4 bye hadoop	world 1	
...	hello 1	
	hadoop 1	
	bye 1	
	hadoop 1	

MapReduce : WordCount

INPUT	MAP	REDUCE
#1 hello world	hello 1	
#2 goodbye world	world 1	
#3 hello hadoop	goodbye 1	
#4 bye hadoop	world 1	
...	hello 1	
	hadoop 1	
	bye 1	
	hadoop 1	

MapReduce : WordCount

INPUT	MAP	REDUCE
#1 hello world	hello 1	hello 2
#2 goodbye world	world 1	
#3 hello hadoop	goodbye 1	
#4 bye hadoop	world 1	
...	hello 1	
	hadoop 1	
	bye 1	
	hadoop 1	

MapReduce : WordCount

INPUT	MAP	REDUCE
#1 hello world	hello 1	hello 2
#2 goodbye world	world 1	
#3 hello hadoop	goodbye 1	
#4 bye hadoop	world 1	
...	hello 1	
	hadoop 1	
	bye 1	
	hadoop 1	

MapReduce : WordCount

INPUT	MAP	REDUCE
#1 hello world	hello 1	hello 2
#2 goodbye world	world 1	world 2
#3 hello hadoop	goodbye 1	
#4 bye hadoop	world 1	
...	hello 1	
	hadoop 1	
	bye 1	
	hadoop 1	

MapReduce : WordCount

INPUT	MAP	REDUCE
#1 hello world	hello 1	hello 2
#2 goodbye world	world 1	world 2
#3 hello hadoop	goodbye 1	
#4 bye hadoop	world 1	
...	hello 1	
	hadoop 1	
	bye 1	
	hadoop 1	

MapReduce : WordCount

INPUT	MAP	REDUCE
#1 hello world	hello 1	hello 2
#2 goodbye world	world 1	world 2
#3 hello hadoop	goodbye 1	goodbye 1
#4 bye hadoop	world 1	
...	hello 1	
	hadoop 1	
	bye 1	
	hadoop 1	

MapReduce : WordCount

INPUT	MAP	REDUCE
#1 hello world	hello 1	hello 2
#2 goodbye world	world 1	world 2
#3 hello hadoop	goodbye 1	goodbye 1
#4 bye hadoop	world 1	
...	hello 1	
	hadoop 1	
	bye 1	
	hadoop 1	

MapReduce : WordCount

INPUT	MAP	REDUCE
#1 hello world	hello 1	hello 2
#2 goodbye world	world 1	world 2
#3 hello hadoop	goodbye 1	goodbye 1
#4 bye hadoop	world 1	hadoop 2
...	hello 1	
	hadoop 1	
	bye 1	
	hadoop 1	

MapReduce : WordCount

INPUT	MAP	REDUCE
#1 hello world	hello 1	hello 2
#2 goodbye world	world 1	world 2
#3 hello hadoop	goodbye 1	goodbye 1
#4 bye hadoop	world 1	hadoop 2
...	hello 1	
	hadoop 1	
	bye 1	
	hadoop 1	

MapReduce : WordCount

INPUT	MAP	REDUCE
#1 hello world	hello 1	hello 2
#2 goodbye world	world 1	world 2
#3 hello hadoop	goodbye 1	goodbye 1
#4 bye hadoop	world 1	hadoop 2
...	hello 1	bye 1
	hadoop 1	
	bye 1	
	hadoop 1	

Hadoop

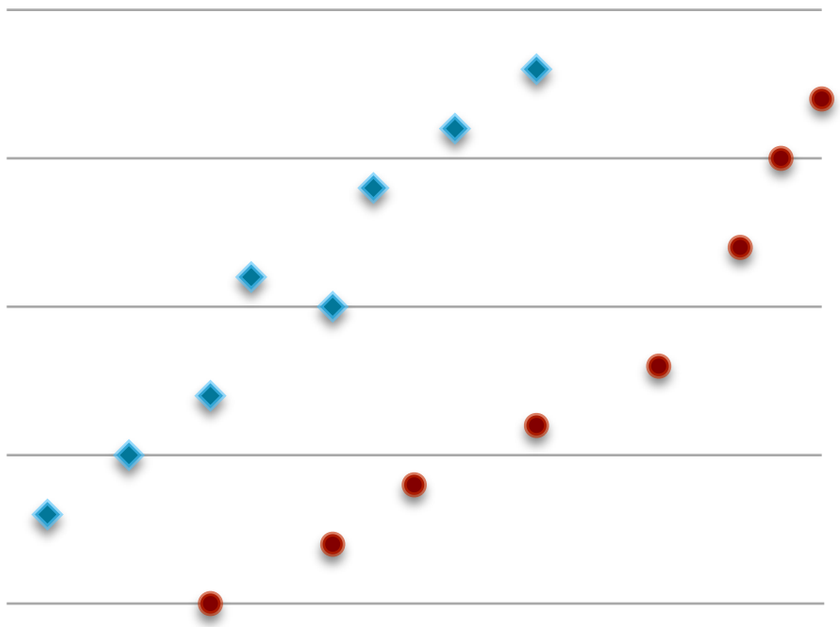
- Apache's implementation of MapReduce
- Consists of
  - Distributed **storage system** : HDFS
  - **Execution** framework : Hadoop MapReduce
- Master node which orchestrates the task distribution
- Worker nodes which perform the tasks
  - Typical node runs a **DataNode** and **TaskTracker**

Support Vector Machines

- Computes a decision boundary (**hyperplane**) that separates inputs of different **classes** represented in a given **feature space** transformed by a given kernel
- The values of two parameters need to be adapted to the data:
  - Cost **C** of errors
  - **σ** of the Gaussian kernel

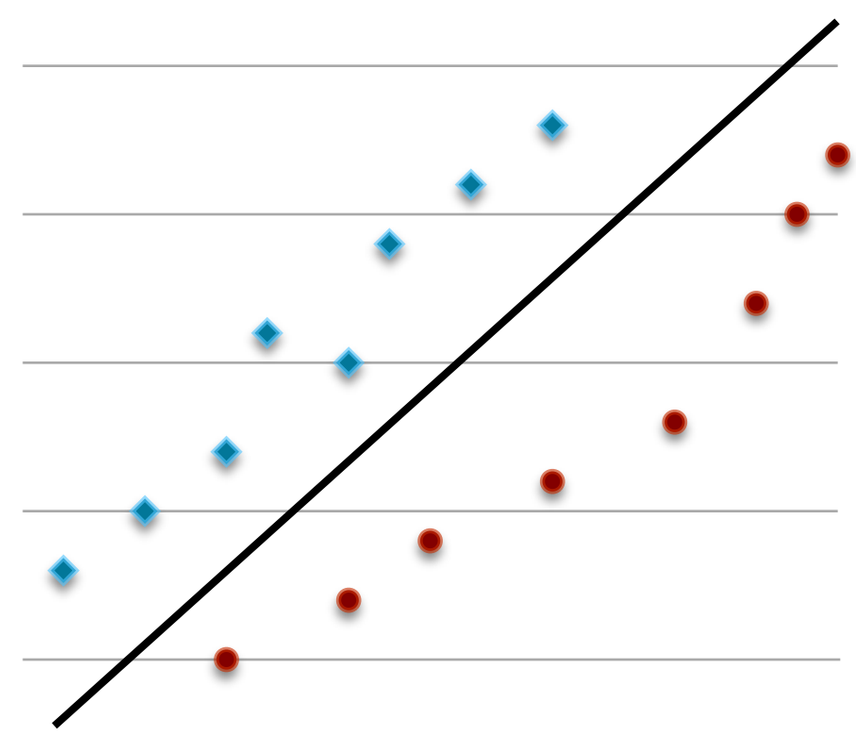
Support Vector Machines

- Computes a decision boundary (**hyperplane**) that separates inputs of different **classes** represented in a given **feature space** transformed by a given kernel
- The values of two parameters need to be adapted to the data:
  - Cost **C** of errors
  - **σ** of the Gaussian kernel



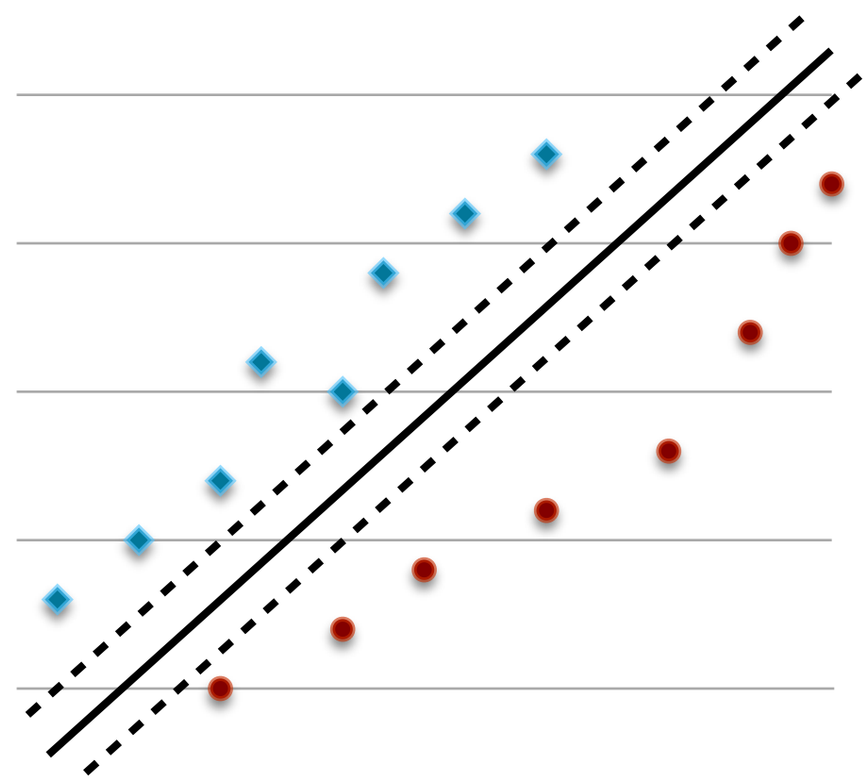
Support Vector Machines

- Computes a decision boundary (**hyperplane**) that separates inputs of different **classes** represented in a given **feature space** transformed by a given kernel
- The values of two parameters need to be adapted to the data:
  - Cost **C** of errors
  - **σ** of the Gaussian kernel



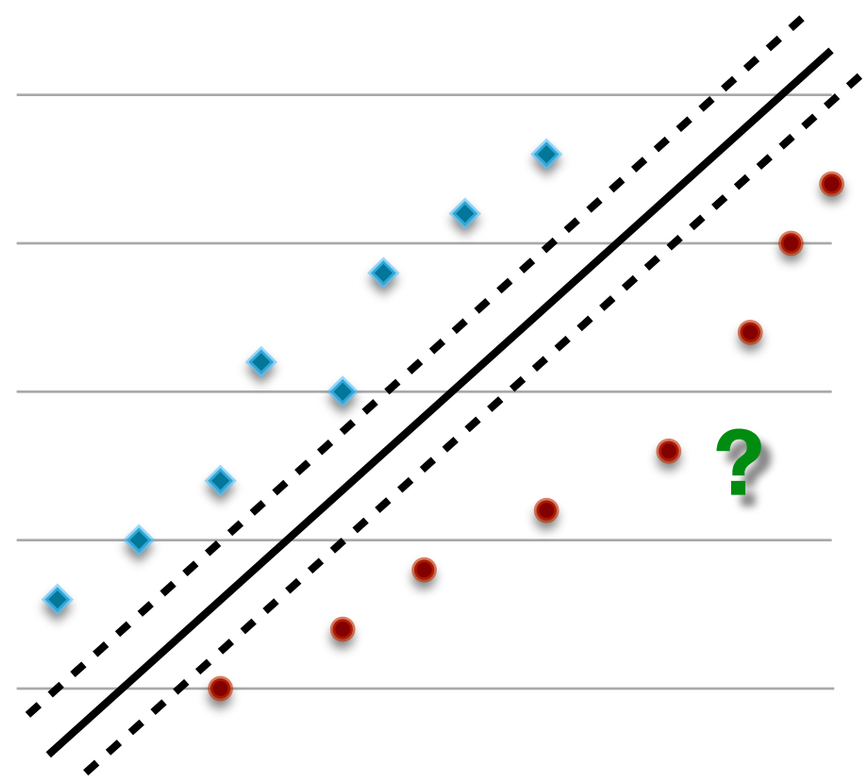
Support Vector Machines

- Computes a decision boundary (**hyperplane**) that separates inputs of different **classes** represented in a given **feature space** transformed by a given kernel
- The values of two parameters need to be adapted to the data:
  - Cost **C** of errors
  - **σ** of the Gaussian kernel



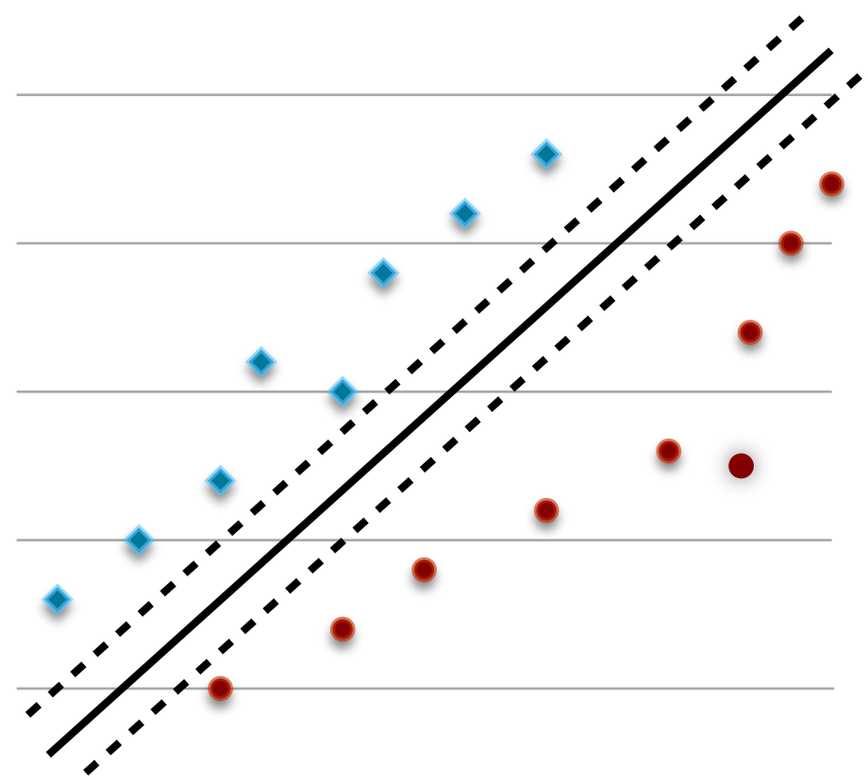
Support Vector Machines

- Computes a decision boundary (**hyperplane**) that separates inputs of different **classes** represented in a given **feature space** transformed by a given kernel
- The values of two parameters need to be adapted to the data:
  - Cost **C** of errors
  - **σ** of the Gaussian kernel



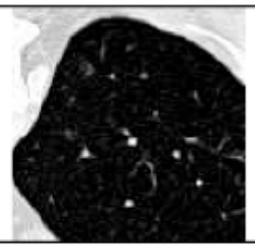
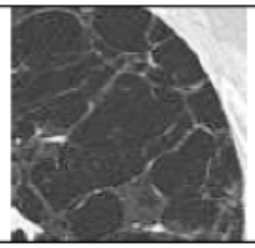
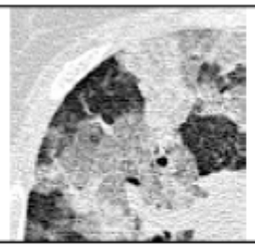
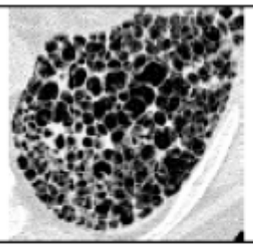
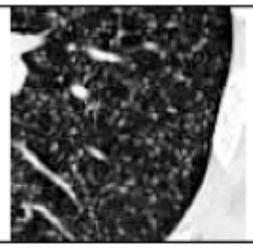
Support Vector Machines

- Computes a decision boundary (**hyperplane**) that separates inputs of different **classes** represented in a given **feature space** transformed by a given kernel
- The values of two parameters need to be adapted to the data:
  - Cost **C** of errors
  - **σ** of the Gaussian kernel



SVM (continued)

- Goal : find optimal value couple (C, σ) to train a SVM
  - Allowing best classification performance of 5 lung texture patterns

visual aspect					
class	healthy	emphysema	ground glass	fibrosis	micronodules

- Execution on 1 PC (without Hadoop) can take weeks
  - Due to extensive **leave-one-patient-out** cross-validation with 86 patients
- Parallelization : Split job by parameter value couples

Image Indexing

- Two phases
  - **Extract features** from images
  - Construct **bags of visual words** by quantization
- Component-based / Monolithic approaches
- Parallelization : Each task processes *N* images

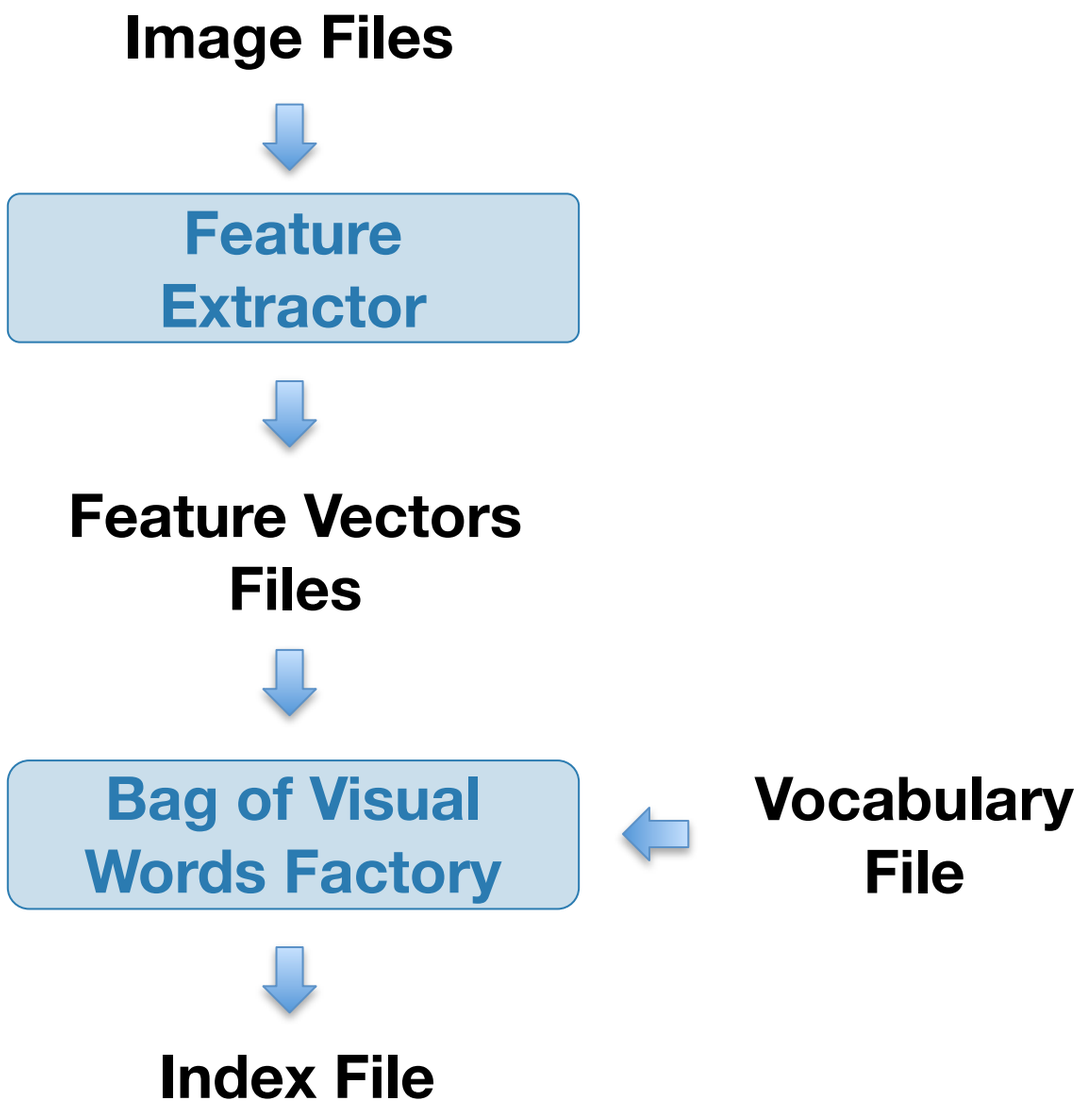
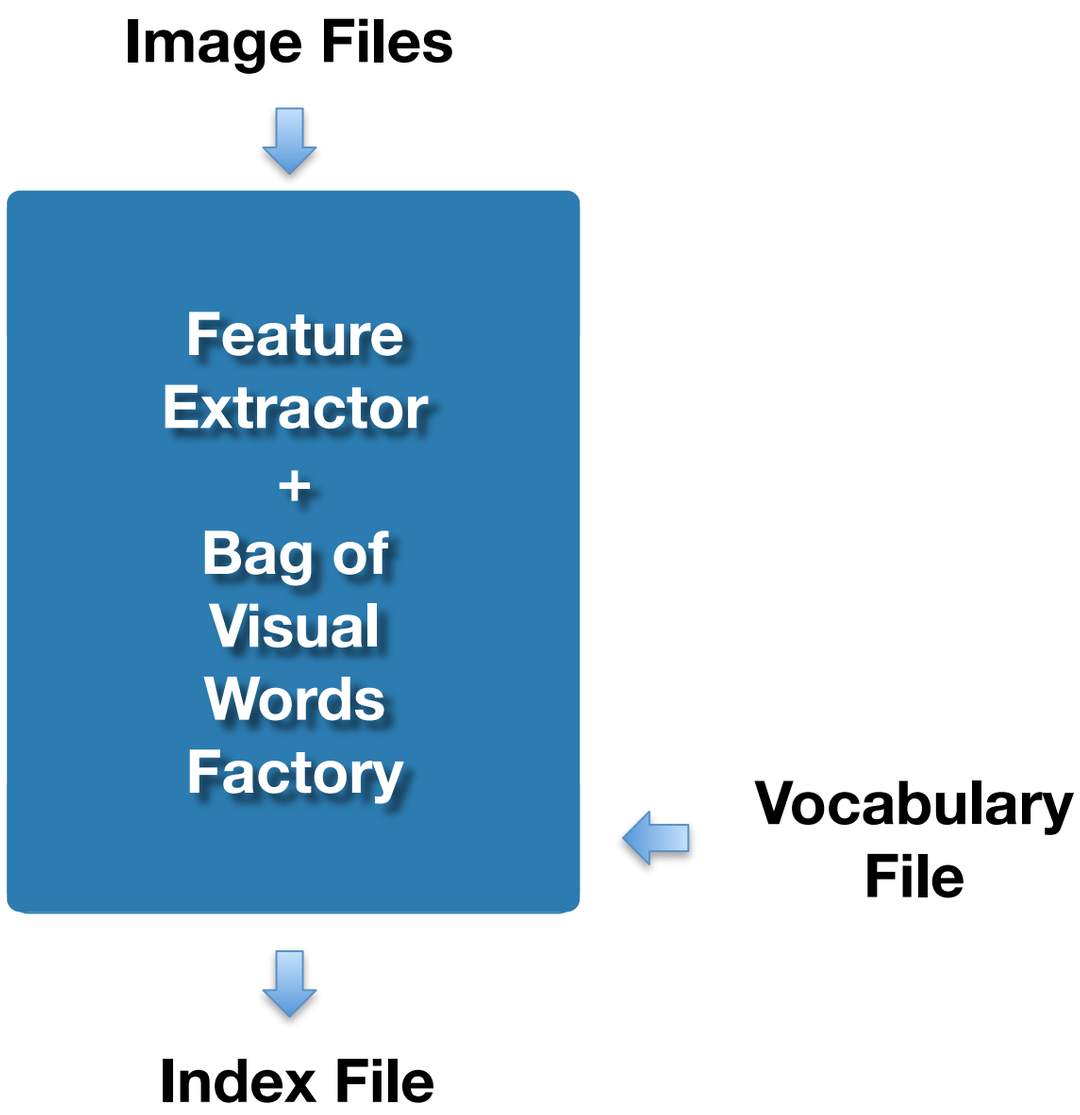


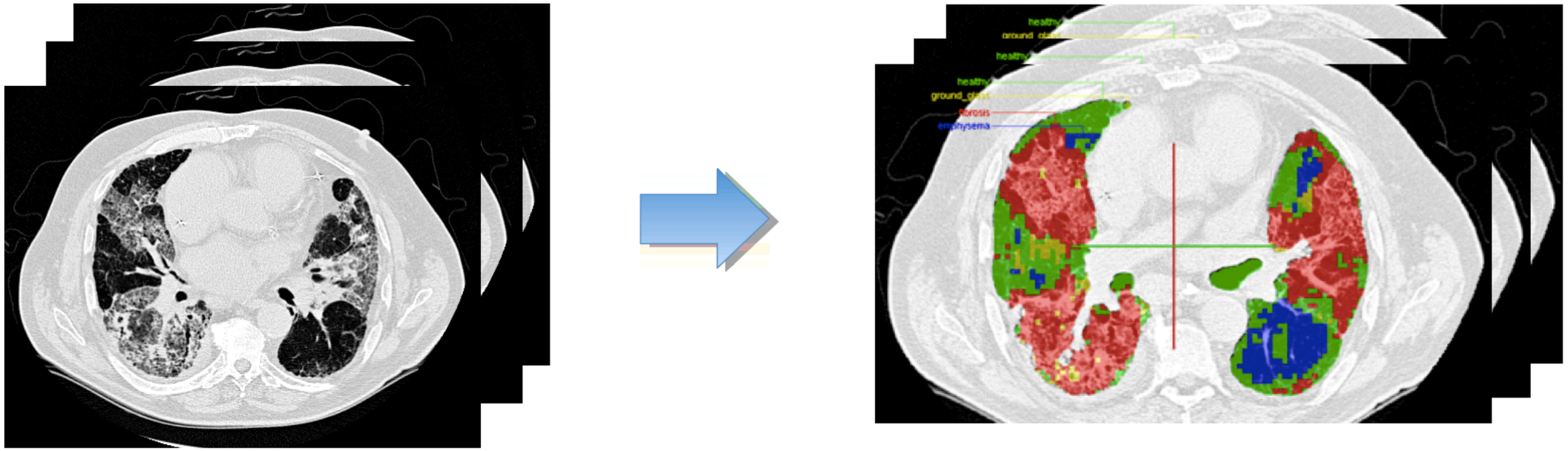
Image Indexing

- Two phases
  - **Extract features** from images
  - Construct **bags of visual words** by quantization
- Component-based / Monolithic approaches
- Parallelization : Each task processes *N* images



3D Texture Analysis (Riesz)

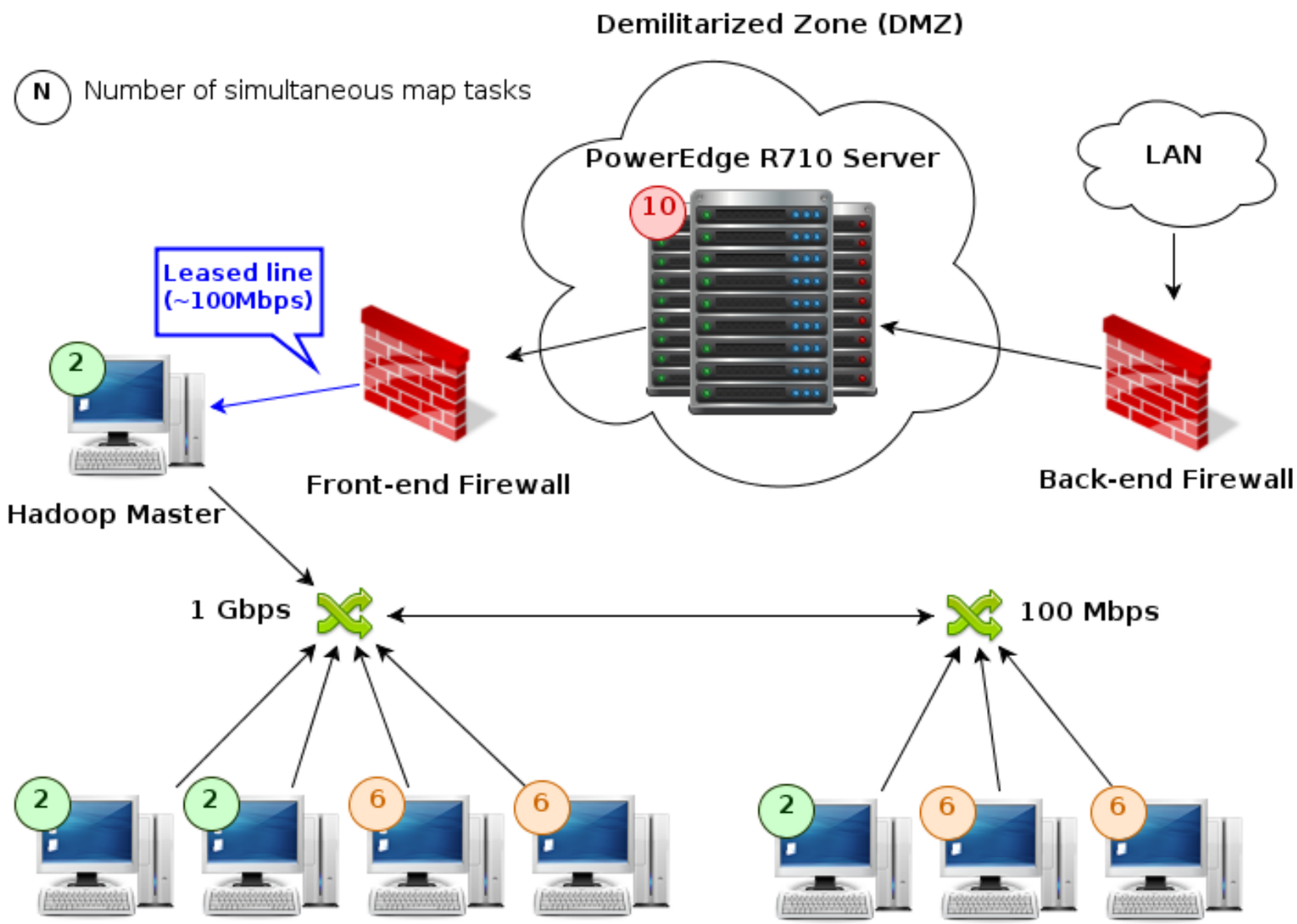
- Features are extracted from 3D images (see below)



- Parallelization : Each task processes *N* images

Results & Interpretation

Hadoop Cluster

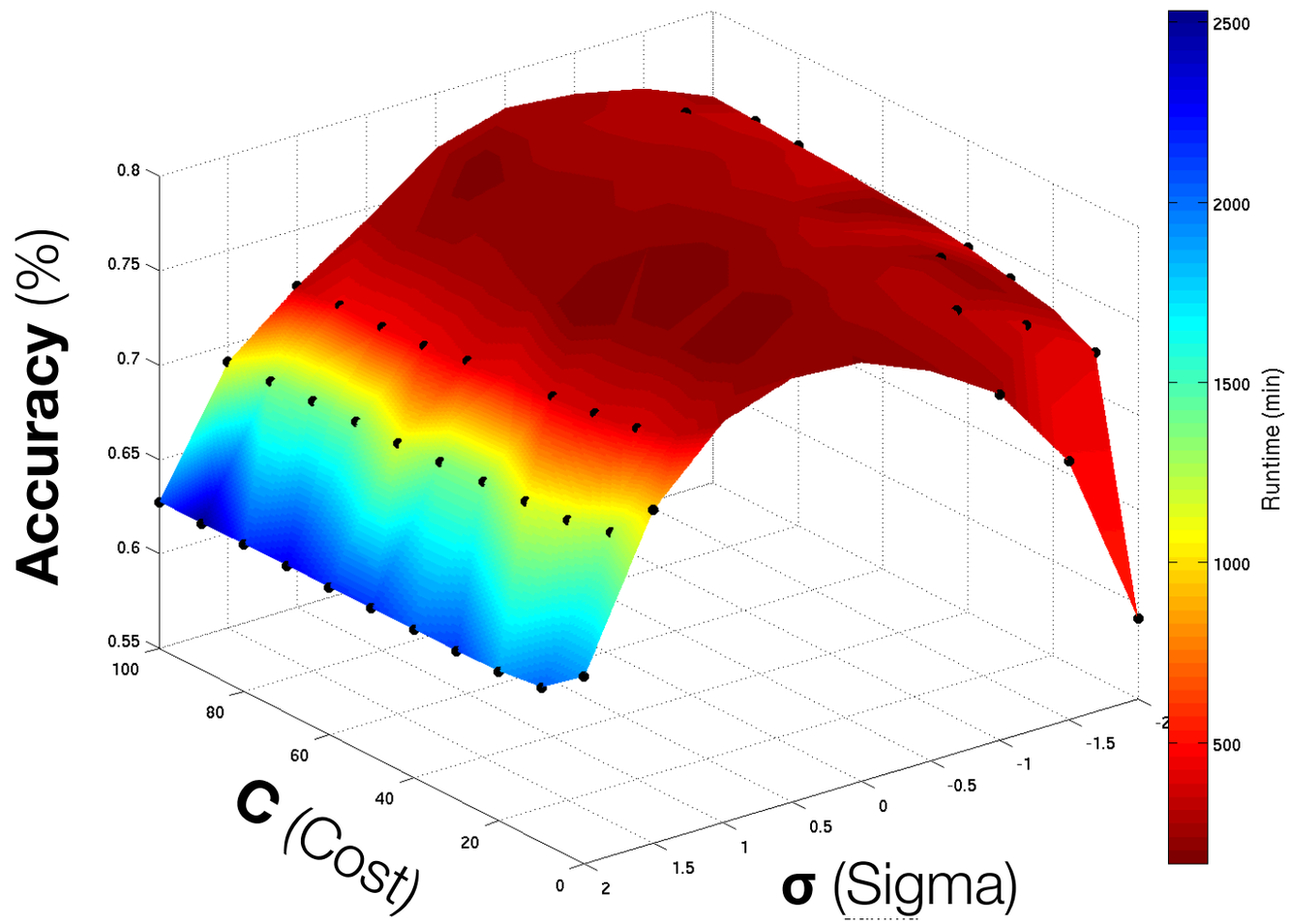


- Minimally invasive setup (>=2 free cores per node)

Support Vector Machines

- Optimization : **Longer tasks = bad performance**
  - Because the optimization of the hyperplane is more difficult to compute (more iterations needed)
  - After 2 patients (out of 86), check if : $t_{\mathrm{i}} \geq F \cdot t_{\mathrm{ref}}$.
  - If time exceeds average (+margin), **terminate** task

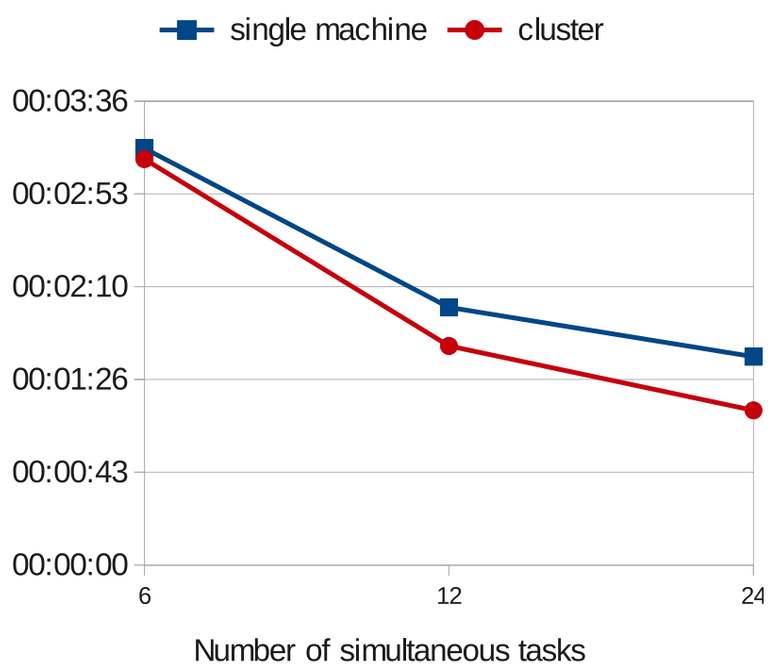
Support Vector Machines



- Black ● : tasks to be interrupted by the new algorithm
- Optimized algorithm : ~50h → ~9h15min
  - All the best tasks (highest accuracy) are **not killed**

Image Indexing

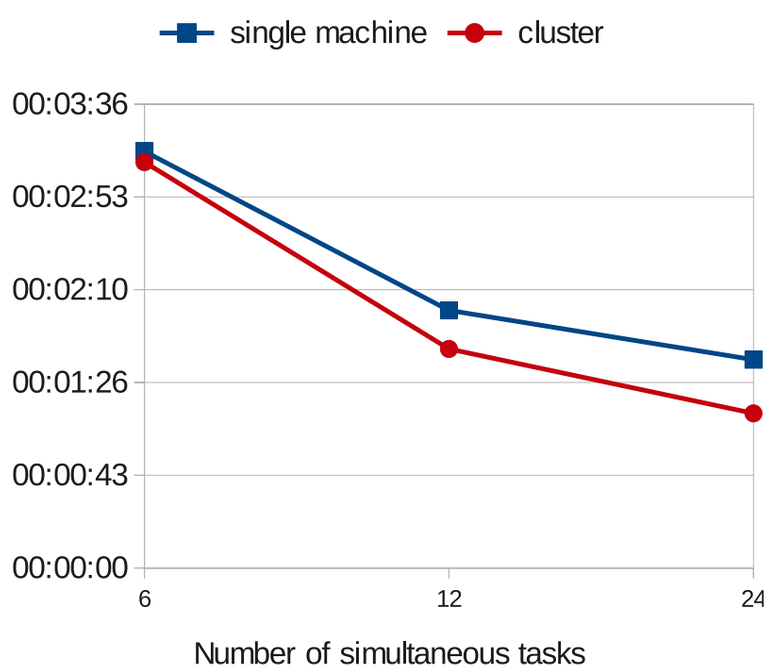
1K IMAGES



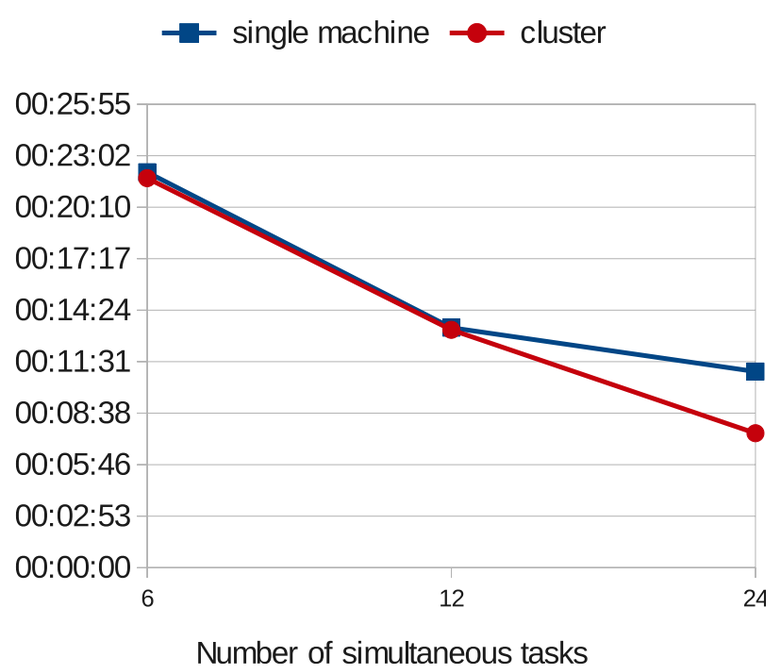
- Shows the calculation time in function of the # of tasks
- Both experiments were executed using hadoop
  - Once on a single computer, then on our cluster of PCs

Image Indexing

1K IMAGES



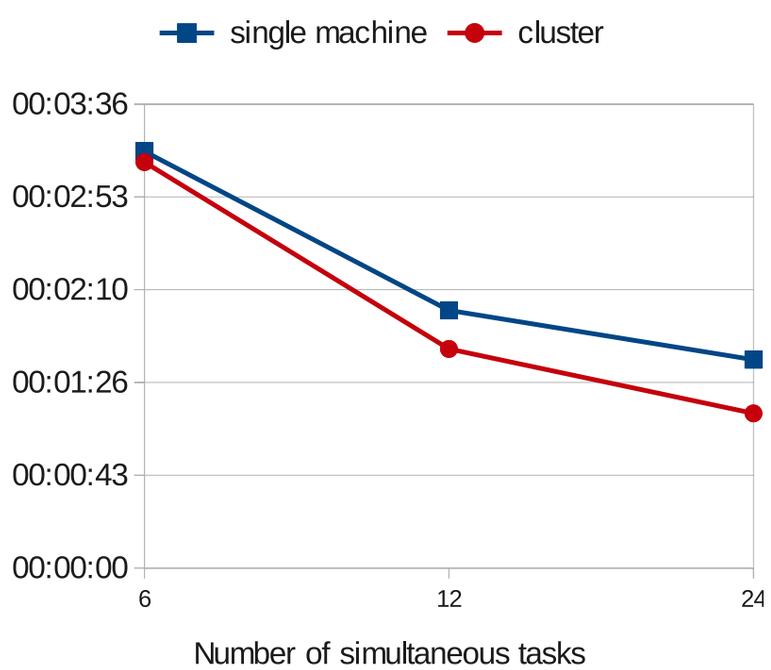
10K IMAGES



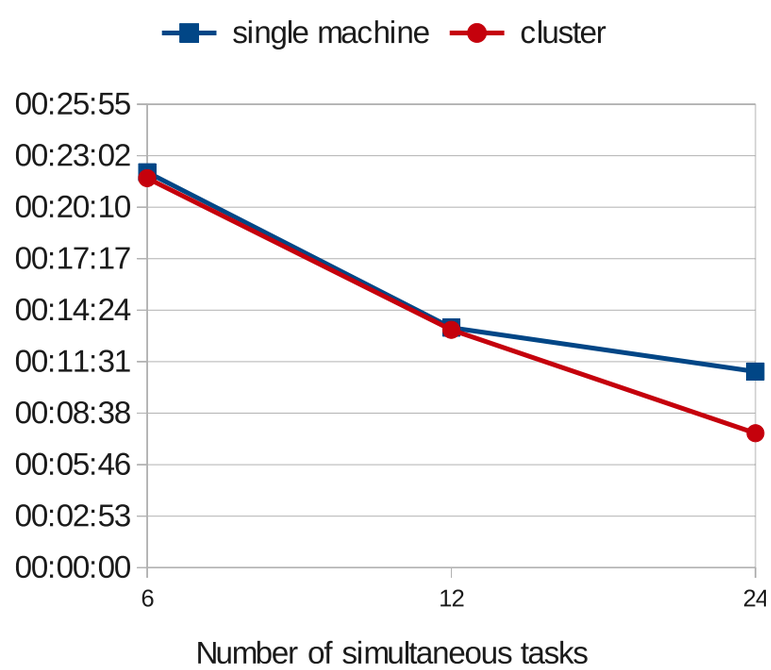
- Shows the calculation time in function of the # of tasks
- Both experiments were executed using hadoop
  - Once on a single computer, then on our cluster of PCs

Image Indexing

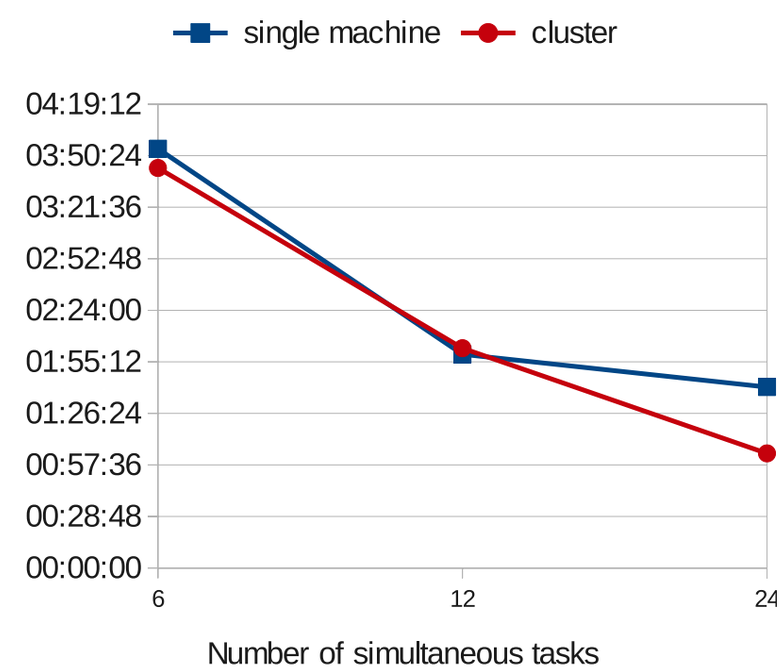
1K IMAGES



10K IMAGES



100K IMAGES



- Shows the calculation time in function of the # of tasks
- Both experiments were executed using hadoop
  - Once on a single computer, then on our cluster of PCs

Riesz 3D

- Particularity : code was a series of Matlab® scripts
- Instead of rewriting the whole application :
  - Used Hadoop **streaming** feature (uses stdin/stdout)
  - To maximize scalability, GNU Octave was used
  - Great compatibility between Matlab® and Octave

Riesz 3D

- Particularity : code was a series of Matlab® scripts
- Instead of rewriting the whole application :
  - Used Hadoop **streaming** feature (uses stdin/stdout)
  - To maximize scalability, GNU Octave was used
  - Great compatibility between Matlab® and Octave

Results

1 task (no Hadoop)	42 tasks (idle)	42 tasks (normal)
131h32m42s	**6h29m51s**	**5h51m31s**

Conclusions

Conclusions

- MapReduce is
  - Flexible (worked with very varied use cases)
  - Easy to use (2-phase programming model is simple)
  - Efficient (>=20x speedup for all use cases)
- Hadoop is
  - Easy to deploy & manage
  - User-friendly (nice Web UIs)

Conclusions (continued)

- Speedups for the different use cases

	SVMs	Image Indexing	3D Feature Extraction
Single task	990h*	21h*	131h30
42 tasks on hadoop	50h / 9h15**	1h	5h50
Speedup	**20x / 107x****	**21x**	**22.5x**

* estimation ** using the optimized algorithm

Lessons Learned

- It is important to use **physically distributed** resources
  - Overloading a single machine hurts performance
  - **Data locality** notably speeds up jobs
- Not every application is infinitely scalable
  - Performance improvements level off at some point

Future work

- Take it to the next level : **The Cloud**
  - Amazon Elastic Cloud Compute (IaaS)
  - Amazon Elastic MapReduce (PaaS)

- Cloudbursting
  - Use both local resources + Cloud (for peak usage)

Thank you ! Questions ?