UNIVERSIDADE DE ÉVORA

Cloud Computing e padrões na computação distribuída

Aplicações em Nuvem

- Cloud computing é interessante para utilizadores:
  - motivos económicos
    - investimento inicial em infraestrutura é reduzido
    - baixo custo de operação (paga-se apenas o que se usa)
  - Conveniência e desempenho
    - *developers* beneficiam com infraestrutura *just-in-time*; podem conceber uma aplicação sem preocupações com a plataforma
    - o potencial de aproveitamento de paralelização pode reduzir o tempo de execução de aplicações *compute-intensive* e/ou *data-intensive*

Cloud computing é também interessante para *Service Providers*:

- oferece condições para uma maior utilização de recursos, mediante a partilha (virtualização e camadas)

Desafios no desenvolvimento de aplicações

- Gestão do desempenho - é complicado oferecer garantias fortes do nível de desempenho, especialmente em situações de maior carga
- Fiabilidade - uma falha nos servidores tem um impacto muito forte; e um sistema que depende de muitos servidores terá certamente a experiência de perder um deles.
- Variabilidade na latência e largura de banda disponível
- Compromisso entre o detalhe de *data logging* e a capacidade de posteriormente identificar causa de erros; e o impacto que a verbosidade dos logs pode ter no desempenho

Possibilidades / abordagens

- Três grandes categorias de aplicação:
  - *Stream processing pipelines*
  - *Batch processing systems*
  - *Web applications*

Contextos em que podem aplicar-se

- *Batch processing* para sistemas de apoio à decisão e analítica
- Aplicações móveis interativas que envolvam muitos dados e redes de sensores
- Aplicações científicas *compute-intensive* e *data-intensive*

Data pipelines

Data pipelines lêem dados de uma fonte, aplicam uma série de operações (filtros, transformações), e emitem o resultado para um repositório

- Batch Data Pipelines
  - Uma execução implica tratar de toda a coleção de dados
  - O tempo de execução depende do volume de dados
- Streaming Data Pipelines
  - Execução contínua
  - Recebem dados de um canal (stream), aplicam operações (filtros, transformações), e emitem o resultado para outro canal (stream)
  - Os dados não estão todos disponíveis ao mesmo tempo!

Apontamentos de Arquitetura para aplicações em cloud

UNIVERSIDADE DE ÉVORA

- Baseado no modelo cliente-servidor
- *Stateless servers* - cada pedido de cliente é tratado de modo independente, não sendo necessário manter um estado ou o prévio estabelecimento de ligação
- Opções para a Comunicação
  - Remote Procedure Calls (RPCs)
  - Simple Object Access Protocol (SOAP) - web applications; message format based on the XML; JSON; protocolos TCP ou UDP
  - Representational State Transfer (REST) - permite comunicação com *stateless servers*, é independente de plataforma e linguagens de programação, e o respetivo tráfego não cria dificuldades com firewalls

Padrões para o Fluxo de execução distribuída (*workflow patterns*)

UNIVERSIDADE DE ÉVORA

- Workflow patterns - *the temporal relationship among the tasks of a process*
  - a) Sequence - several tasks have to be scheduled one after the completion of the other.
  - b) AND split - both tasks B and C are activated when task A terminates.
  - c) Synchronization - task C can only start after tasks A and B terminate.
  - d) XOR split - after completion of task A, either B or C can be activated.
  - e) XOR merge - task C is enabled when either A or B terminate.
  - f) OR split - after completion of task A one could activate either B, C, or both.
  - g) Multiple Merge - once task A terminates, B and C execute concurrently; when the first of them, say B, terminates, then D is activated; then, when C terminates, D is activated again.

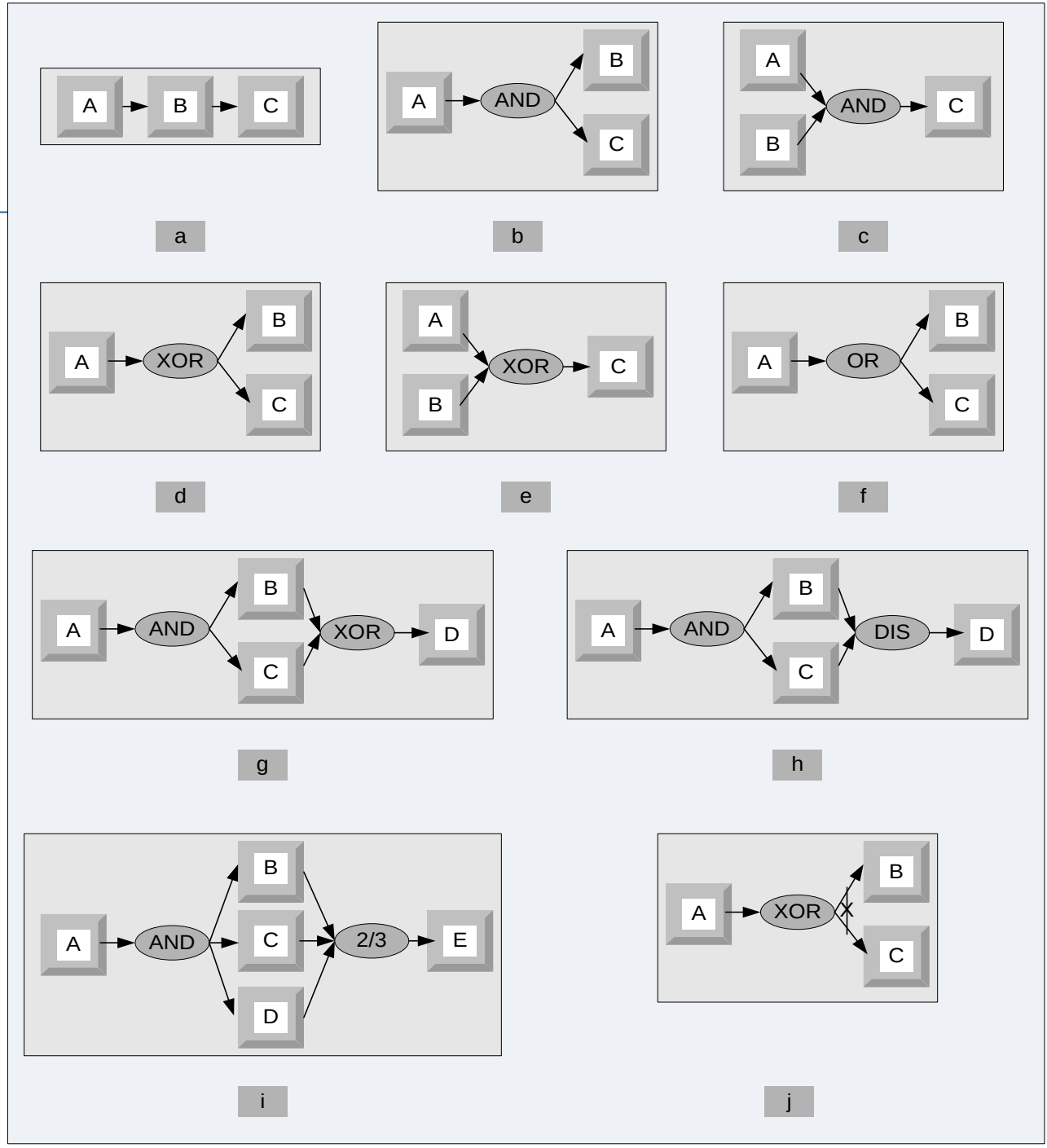
Padrões para o Fluxo de execução distribuída (*workflow patterns*)

UNIVERSIDADE DE ÉVORA

h) Discriminator – wait for a number of incoming branches to complete before activating the subsequent activity; then wait for the remaining branches to finish without taking any action until all of them have terminated.
Next, resets itself.

i) N out of M join - barrier synchronization. Assuming that M tasks run concurrently, N (N<M) of them have to reach the barrier before the next task is enabled. In our example, any two out of the three tasks A, B, and C have to finish before E is enabled.

j) Deferred Choice - similar to the XOR split but the choice is not made explicitly; the run-time environment decides what branch to take.



Dan C. Marinescu

Elasticidade e distribuição de carga

- Elasticidade
  - capacidade de usar tantos recursos quantos os necessários para responder de forma ótima às restrições ou necessidades da aplicação, considerando o tempo e o custo

- Distribuição de Carga

•um servidor de *front-end* distribui os pedidos entre um certo número de servidores de *back-end*

•à medida que a carga aumenta, novos servidores de *back-end* podem ser adicionados à pool do serviço

Aborgagens no particionamento do trabalho computacional

- divisão modular → forma de divisão do trabalho conhecida de antemão
- divisão arbitrária → o trabalho pode ser dividido num nº arbitrariamente grande de pequenas tarefas, de volume igual ou aproximado (divisão dinâmica)

UNIVERSIDADE DE ÉVORA

Recapitulando… com Computação em Nuvem

- Temos:
  - Grande infraestrutura
  - Acesso *on-demand* (*self service*, na hora, *pay-as-you-go*)
  - Processos com uso intensivo de dados
  - Novos paradigmas de programação com distribuição
    - Map-Reduce/Hadoop
    - NoSQL, Cassandra, MongoDB...

UNIVERSIDADE DE ÉVORA

Programar execução em paralelo

É um desafio potencialmente complexo

- Dividir dados, ou subdividir as tarefas
- Afetar “trabalhadores”/unidades de processamento às tarefas
- Gerir o trabalho de cada “trabalhador” sobre cada tarefa
- Monitorizar finalização de tarefas ou erros
- Reiniciar execução de casos que deram erro
- Recolher resultados parciais… e determinar resultado global

UNIVERSIDADE DE ÉVORA

Abordagem: abstração para o processo

- Organizar os dados em blocos mais pequenos, independentes, os Data Chunks (shards?)
- Determinar o trabalho a realizar sobre cada Chunck
- Ter um gestor (*Master*)
  - Divide os dados pelos trabalhadores
  - Recolhe cada resultado parcial
- Trabalhador (*Worker*)
  - Elemento com capacidade de processamento
  - Recebe um chunk de dados
  - Executa uma tarefa sobre esses dados
  - Transmite o resultado ao *Master*

MapReduce

Framework para computação paralela distribuída

Criada pela Google em 2004

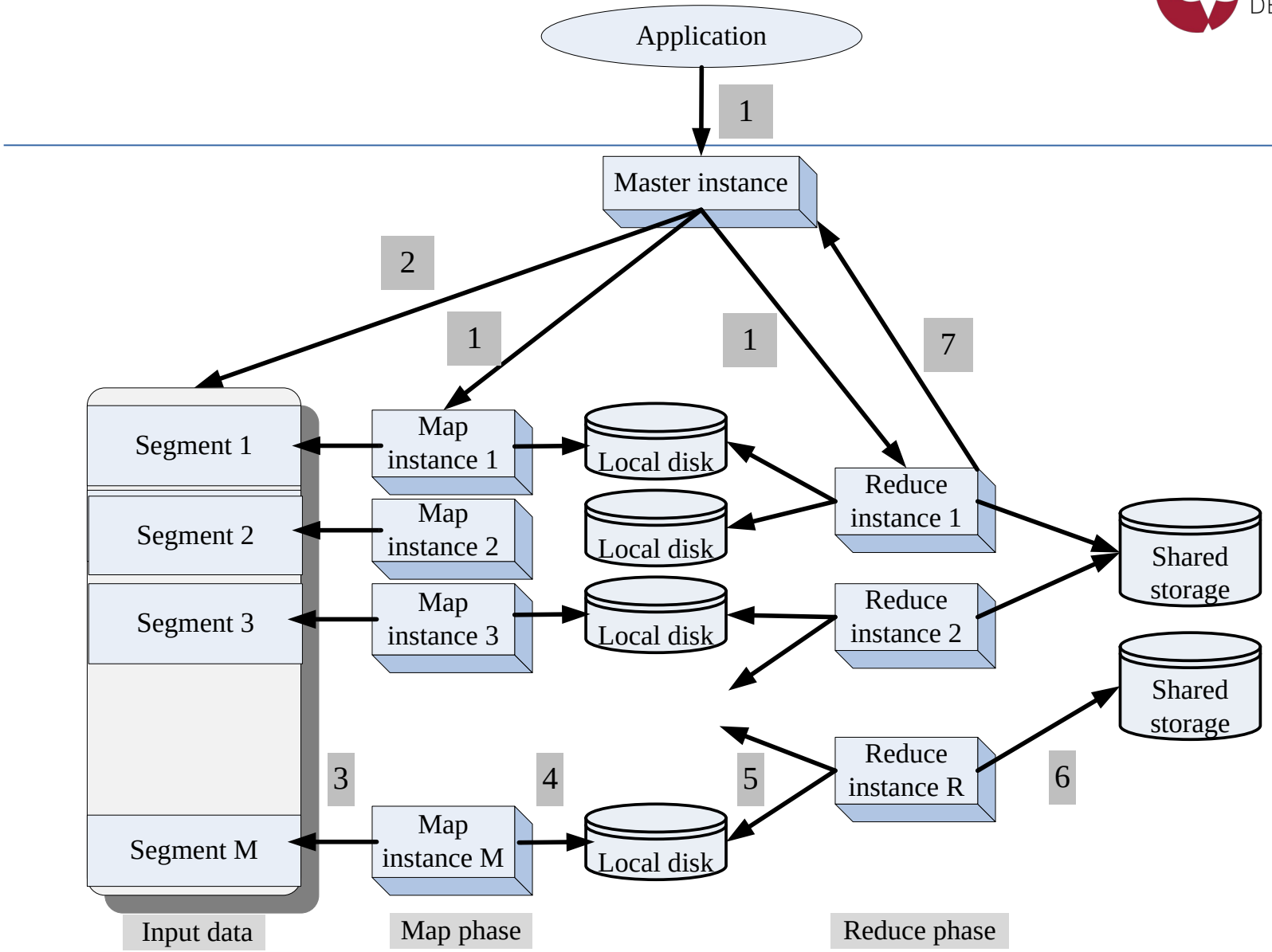
API simples

Inspirado nos pares de funções map() e reduce(), de LISP

Abordagem seguida por muitos gigantes de serviços Cloud para processamento de terabytes e petabytes de dados

MapReduce philosophy

1. An application starts a master instance, M worker instances for the *Map phase* and later R worker instances for the *Reduce phase*.
2. The master instance partitions the input data in M *segments*.
3. Each *map instance* reads its input data segment and processes the data.
4. The results of the processing are stored on the local disks of the servers where the map instances run.
5. When all map instances have finished processing their data, the R reduce instances read the results of the first phase and merge the partial results.
6. The final results are written by the reduce instances to a shared storage server.
7. The master instance monitors the reduce instances and when all of them report task completion the application is terminated.


Application
1
Master instance
2
1
1
7
Segment 1
Segment 2
Segment 3
Segment M
Map instance 1
Map instance 2
Map instance 3
Map instance M
Local disk
Local disk
Local disk
Local disk
Reduce instance 1
Reduce instance 2
Reduce instance R
Shared storage
Shared storage
3
4
5
6
Input data
Map phase
Reduce phase

UNIVERSIDADE DE ÉVORA

MapReduce philosophy

- Inspired by map and reduce primitives in functional programming
  - mapping a function `f` over a sequence `x y z`
    - yields `f(x) f(y) f(z)`
  - reduce combines sequence of elements using a binary `op`

- Many data analysis computations can be expressed as
  - applying a map operation to each logical input record
  - produce a set of intermediate `(key, value)` pairs
  - applying a reduce to all intermediate pairs with same key

Recordar como eram/são as funções map() e reduce() **nas** linguagens funcionais

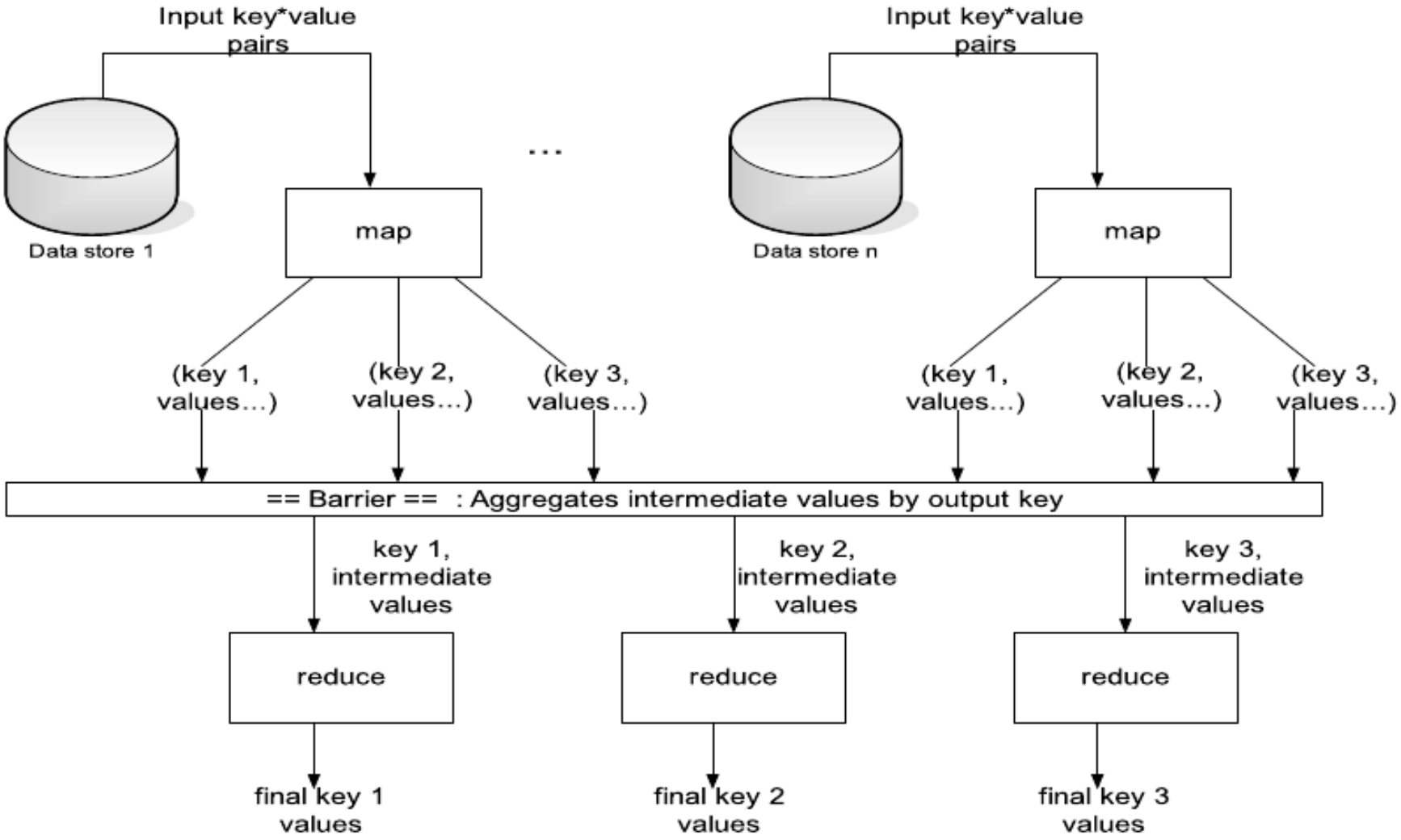
Objetivo: somar os quadrados de uma sequência de inteiros

Duas fases:

- (map square ‘(1 2 3 4))
  - Output: (1 4 9 16)
  - [processa um registo de cada vez, e de modo independente dos demais]
- (reduce + ‘(1 4 9 16))
  - (+ 16 (+ 9 (+ 4 1)))
  - Output: 30
  - [processa o conjunto de todos os registos em grupos, usando um operador +]

UNIVERSIDADE DE ÉVORA

MapReduce: logical view of execution



MR Case study: GrepTheWeb

- The application illustrates the means to
  - create an on-demand infrastructure.
  - run it on a massively distributed system in a manner that allows it to run in parallel and scale up and down, based on the number of users and the problem size.

- GrepTheWeb
  - Performs a search of a very large set of records to identify records that satisfy a regular expression.
  - It is analogous to the Unix *grep* command.
  - The source is a collection of document URLs produced by the Alexa Web Search, a software system that crawls the web every night.
  - Uses message passing to trigger the activities of multiple controller threads which launch the application, initiate processing, shutdown the system, and create billing records.

MapReduce: example data flow after map phase

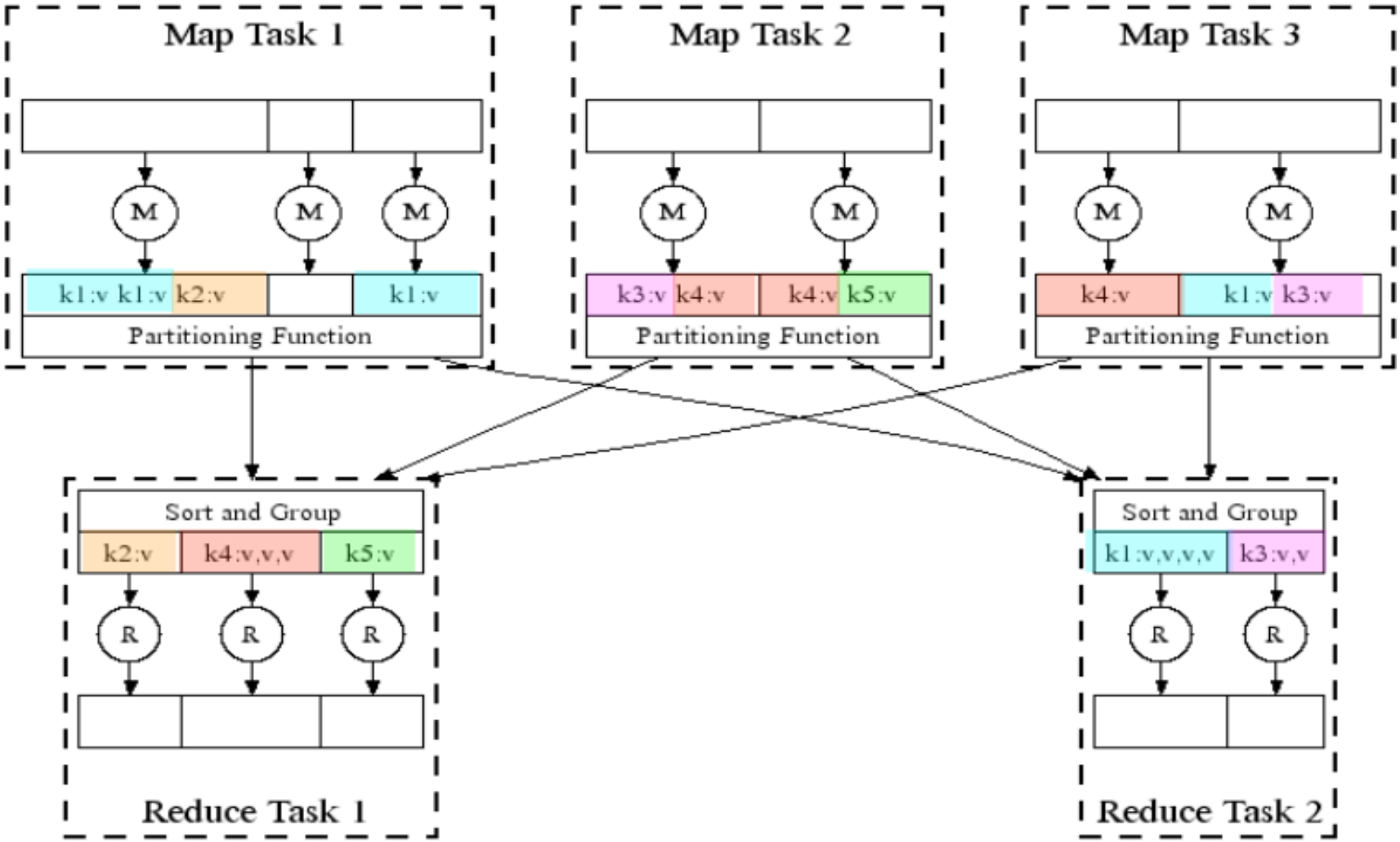
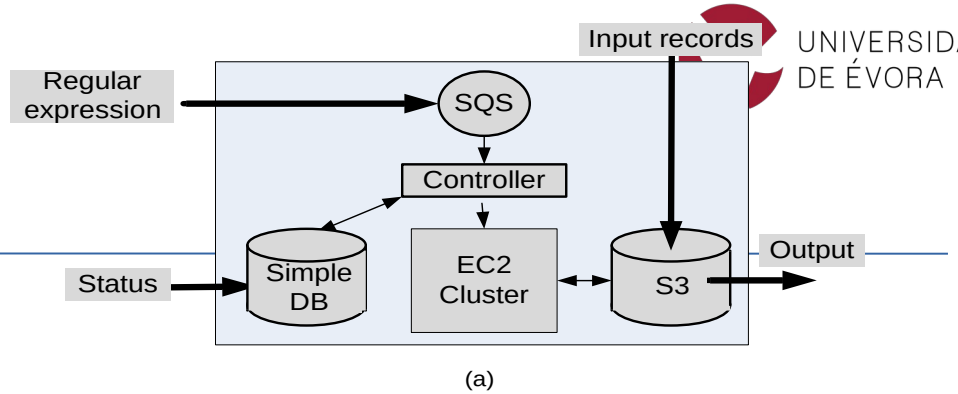


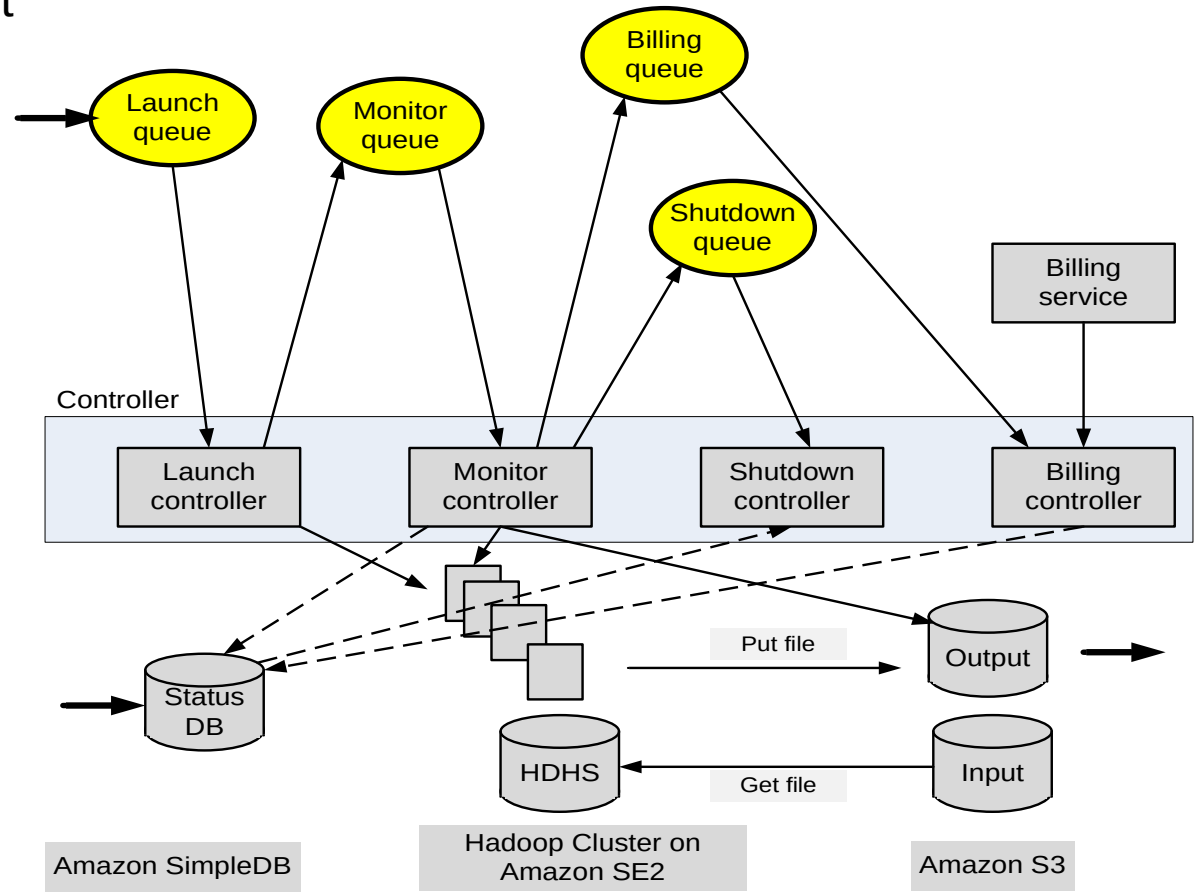
Figure credit: http://labs.google.com/papers/mapreduce-osdi04-slides/index-auto-0008.html

(a) The simplified workflow showing the inputs:
- the regular expression.
- the input records generated by the web crawler.
- the user commands to report the current status and to terminate the processing.



(a)

(b) The detailed workflow. The system is based on message passing between several queues; four controller threads periodically poll their associated input queues, retrieve messages, and carry out the required actions



(b)

Apache Hadoop

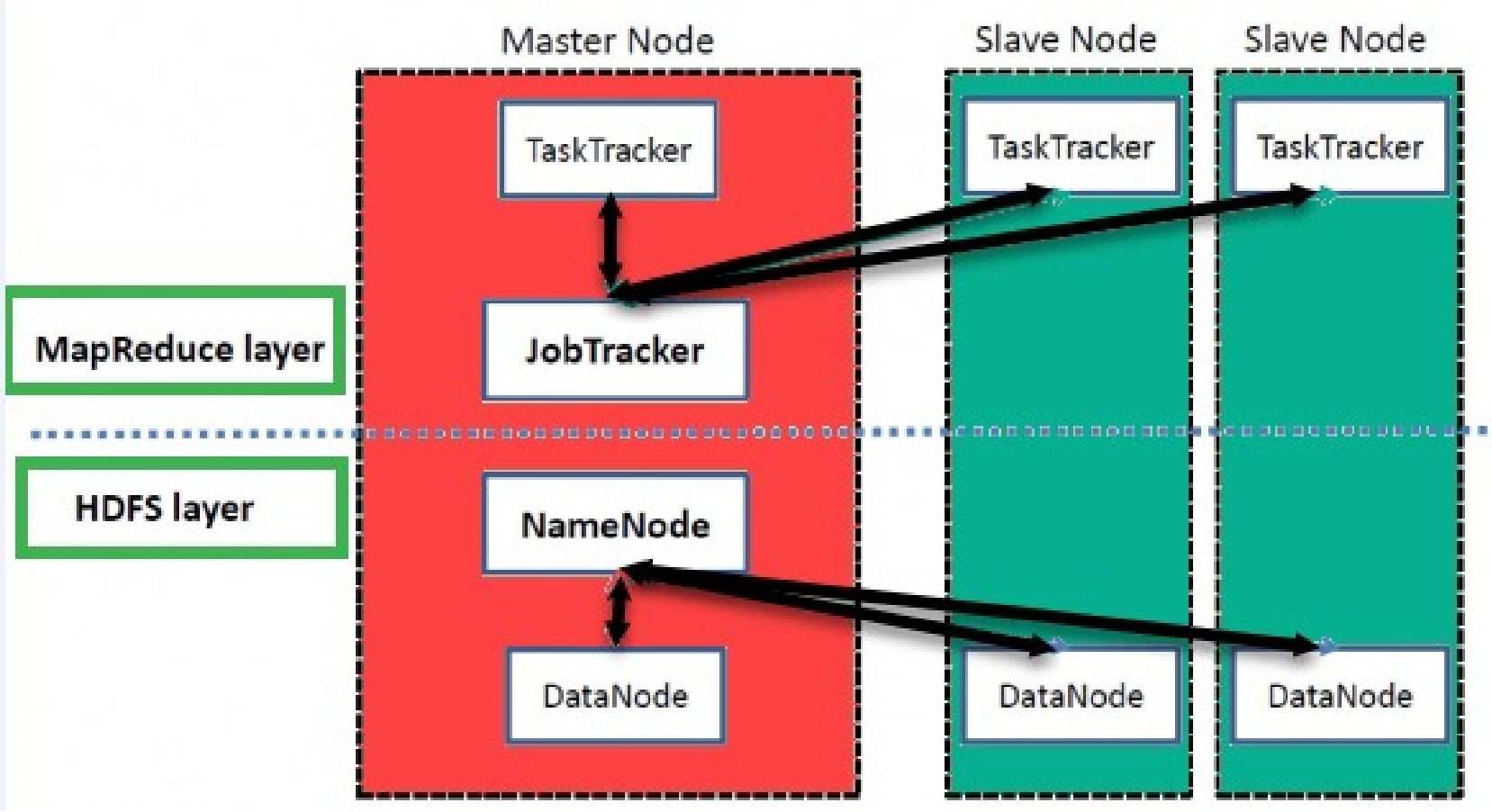
- framework distributed processing of large data sets, across clusters of computers, using simple programming models
  - An open-source implementation of MapReduce

- is an ecosystem composed of modules for computing, storage and coordination in a distributed system:
  - Hadoop Common: The common utilities that support the other Hadoop modules.
  - Hadoop Distributed File System (HDFS™): A distributed file system that provides high-throughput access to application data.
  - Hadoop YARN: A framework for job scheduling and cluster resource management.
  - Hadoop MapReduce: A YARN-based system for parallel processing of large data sets.
  - Other Hadoop-related projects:
    - ZooKeeper, Pig, Hive, Cassandra...

http://hadoop.apache.org/

Apache Hadoop

Visão abstrata da arquitetura de Hadoop



https://www.guru99.com/learn-hadoop-in-10-minutes.html

UNIVERSIDADE DE ÉVORA

Apache Hadoop: excerto de programa (Java)


```
public class WordCount {
public static class TokenizerMapper  extends Mapper<Object, Text, Text, IntWritable> {
    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

    public void map(Object key, Text value, Context context
                ) throws IOException, InterruptedException {
      StringTokenizer itr = new StringTokenizer(value.toString());
      while (itr.hasMoreTokens()) {
        word.set(itr.nextToken());
        context.write(word, one);
      } } }

 public static class IntSumReducer extends
    Reducer<Text,IntWritable,Text,IntWritable>{
    private IntWritable result = new IntWritable();
    public void reduce(Text key, Iterable<IntWritable> values,  Context context
                    ) throws IOException, InterruptedException {
      int sum = 0;
      for (IntWritable val : values) {
        sum += val.get();
      }
      result.set(sum);
      context.write(key, result);
    } }
```

```
public static void main(String[]
  args) throws Exception {

Configuration conf = new
 Configuration();

Job job = Job.getInstance(conf,
 "word count");


 job.setJarByClass(WordCount.
 class);


 job.setMapperClass(Tokenizer
 Mapper.class);
```

https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html

Cloud para ciência e engenharia

- As grandes classes de problema nas diversas áreas:
  - Recolha de dados experimentais
  - Gestão de muito grandes volumes de dados
  - Criar novos modelos e executá-los
  - Integração de dados
  - Documentar experiências realizadas
  - Partilhar dados com outros; preservar dados por muito tempo

- Estas tarefas requerem repositórios “big” data e sistemas compatíveis com muitos ciclos de computação
- O processamento em cloud inclui recursos apropriados e permite ambiente de cooperação na resolução de problemas

Social computing e conteúdos digitais

- Redes permitem a partilha de dados e também a criação de plataformas virtuais para execução remota de tarefas

- ***Volunteer computing*** – utilizadores da internet oferecem tempo de CPU e espaço de armazenamento para apoiar um projeto:
  - Mersenne Prime Search
  - SETI@Home,
  - Folding@home,
  - Storage@Home
  - PlanetLab

Search for **E**xtra**t**errestrial **I**ntelligence (SETI) - You can participate by running a free program that downloads and analyzes radio telescope data. https://setiathome.berkeley.edu (já descontinuado)

UNIVERSIDADE DE ÉVORA

Referências e leituras

- *Cloud Computing: Theory and Practice*

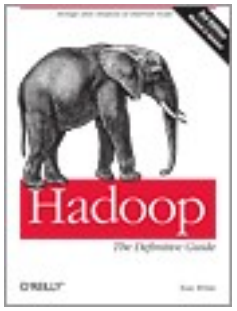
  Dan C. Marinescu

- http://research.google.com/archive/mapreduce.html

- http://hadoop.apache.org/

- Hadoop: The Definitive Guide – 3rd edition

  Tom White. O'Reilly Media



- *Data Analysis with MapReduce*

  John Mellor-Crummey, DCS, Rice University