Disseny i implementació d'un sistema de tractament d'arxius binaris WDF usant MapReduce en clústers Hadoop.

Sergi Pérez Labernia

Arquitectura de computadors i sistemes operatius

Ivan Rodero Castro

28/06/2015

FITXA DEL TREBALL FINAL

Títol del treball:	*Disseny i implementació d'un sistema de tractament d'arxius binaris WDF usant MapReduce en clústers Hadoop.*
Nom de l'autor:	*Sergi Pérez Labernia*
Nom del consultor:	*Ivan Rodero Castro*
Data de lliurament:	*06/2015*
Àrea del Treball Final:	*Arquitectura de computadors i sistemes operatius*
Titulació:	*Grau en Enginyeria Informàtica de computadors*

Resum del Treball:

L'objectiu del projecte es centra en el disseny i la implementació d'un sistema *Big Data* poc convencional capaç d'extreure informació de temps i potència d'arxius binaris. Inicialment, aquests són generats mitjançant un equip d'instrumentació i es tracten repartits entre els diferents nodes. Per a la distribució del treball s'utilitza l'entorn *Hadoop* i la programació en *Java* de les funcions *MapReduce* que permeten guardar de manera distribuïda la informació en el sistema de fitxers *HDFS*. Per acabar, la interfície web permet l'accés a aquesta informació a partir d'una estampa temporal inicial i una de final que s'utilitzen per generar el gràfic lineal d'aquest rang. El sistema resultant és viable i executa el procés en condicions en les que cal afegir control d'errors. La feina es distribueix entre els nodes correctament i evidencia la possibilitat d'ús de *Hadoop* amb fitxers diferents als textuals.

Abstract:

The design and the implementation of non-conventional *Big Data* system are the main aids of this project. This system is able to extract information such as temporal and power values from binary files. Initially, the instrumentation equipment produces binary files which are distributed between all the clustered nodes. The *MapReduce* model allows this system to execute works distributed into Hadoop environment. Furthermore, *Hadoop* saves information between nodes into HDFS file system. On the other hand, the web interface allows to generate graphic outputs ranked between initial and final timestamps. Systems using binary files in Hadoop environment are possible. In future, it would be interesting to work in control errors in order to improve this base system. To conclude, all the computer work is distributed correctly between nodes and the lecture of these binary files demonstrates the possibilities of Hadoop with non-text inputs.

Paraules clau:

Bigdata, Hadoop, Hdfs, MapReduce, Java, Tomcat, JfreeChart, Sistema

Índex

Llista de figures

1. Introducció

Els sistemes de mesura i control van néixer per mantenir el correcte funcionament de diferents processos. Els avenços tecnològics i les millores en les capacitats del maquinari d'instrumentació[1] d'aquests sistemes han donat pas a eines que permeten un control exhaustiu i detallat de cada procés. Amb la millora en el control i la quantitat de les dades resultats els sistemes de tractament tradicionals han deixat de ser factibles.

El projecte *TIC*[2] descrit a continuació per al treball final de grau de l'itinerari d'enginyeria de computadors descriu un sistema concret i escalable per al tractament de grans col·leccions de mesures d'instrumentació. El projecte es situa dins l'àmbit de l'arquitectura de computadors i sistemes operatius. La proposta de treball implica la creació d'un sistema poc convencional de *Big data*[3].

Per començar, es tracten les dades extretes a partir d'equips d'instrumentació per a ser gestionades posteriorment. A continuació, es veuran diferents alternatives per a l'emmagatzematge de les dades tractades mitjançant *Hadoop*[4]. Per acabar, es treballarà en relació amb una interfície[5] d'accés i consulta de la informació. L'elecció de les diferents opcions seran aspectes importants a tenir en compte durant l'elaboració del treball.

1.1. Context

Un conjunt de servidors de la *RDI2*[6] amb certa rellevància treballen sota la supervisió i control d'un aparell d'instrumentació model *DL850*[7] de la marca *Yokogawa*[8]. Aquest genera mesures de potència rebuda per les fonts d'alimentació d'aquests equips cada mil·lèsima part de segon i permet observar en temps real les diferents fluctuacions que van sorgint. Aquest és el context d'un sistema que ha de permetre el tractament, l'emmagatzematge i el posterior visionat per donar major utilitat a la instrumentació dels servidors.

La rellevància d'aquest sistema recau en dos vessants. Per una banda, el cost d'aquest tipus de maquinari d'instrumentació implica l'aplicació sobre processos importants i crítics. Per altre banda, és un sistema aplicable a d'altres processos d'instrumentació amb característiques semblants només fent-ne petites adaptacions.

1.2. Motivació

El marc actual de constant generació de dades i la importància de produir informació útil a partir grans volums d'aquestes dades són factors clau per entendre la necessitat de sistemes que ho gestionin. El disseny i la implementació de sistemes que permeten una gestió adient són necessaris per a l'extracció de conclusions de manera senzilla.

A més, l'enriquidor treball de comparació entre diferents dissenys de sistemes de processament en paral·lel amb *Hadoop* afegeix valor al treball. L'escalabilitat en el sistema d'emmagatzematge serà també important i a tenir en compte. Per acabar, la implementació d'una interfície per accedir a la informació mitjançant un model d'accés propi a *HDFS*[10] aportarà la rúbrica al treball.

1.3. Descripció del projecte

Amb aquest projecte es pretén desenvolupar un sistema no convencional de Big data en el que es podrà extreure informació a partir de dades produïdes per equips d'instrumentació. Es parla d'un sistema o conjunt d'eines de software que es puguin utilitzar en computadors i centres de dades de manera còmoda i escalable. Concretament, en el cas del projecte es tracten mesures de corrent de servidors que es generen a partir de maquinari d'instrumentació *Yokogawa*. Per tant, durant la realització del projecte es tractaran les dades per adaptar-les a les necessitats, es provaran diferents solucions d'emmagatzematge i s'implementarà una solució per accedir adientment i mostrar la informació.

1.4. Objectius

S'enumeren a continuació els principals objectius del projecte:

- Tractament i gestió de dades.
- Veure a la pràctica la diferència entre dades i informació.
- Aprenentatge teòric sobre *Big data*.
- Provar diferents sistemes d'emmagatzematge per a *Big data*.
- Introduir-se en el funcionament de *frameworks*[9] com *Apache Spark*[11] o *Hadoop*.
- Realitzar una implementació de programació paral·lela *MapReduce*[12] amb èxit.
- Discernir entre possibles solucions per a una situació de computació paral·lela.
- Realitzar una proposta d'interfície per accedir de manera còmoda a la informació.
- Documentar en relació a la presa de decisió i els criteris de selecció que s'han seguit per a les diferents solucions.
- Documentar els resultats del procés de manera entenedora per facilitar la reproducció del sistema.

1.5. Enfocament i metodologia

L'estratègia es basa en el desenvolupament del sistema mitjançant un sistema d'un sol node o amb la simulació mitjançant màquines virtuals d'una petita xarxa de nodes i la seva posterior implementació en els sistemes en producció. Aquesta estratègia permet realitzar més proves amb menor dependència de recursos.

1.6. Planificació del treball

Els **recursos temporals** destinats al projecte venen delimitats pel caràcter quadrimestral i el nombre de crèdits essent el còmput aproximat d'unes 225 hores.

Els **recursos materials** tecnològics consten de l'equip d'instrumentació *Yokogawa DL 850* junt amb la màquina amb Windows 7 on aquest està connectat i per a la implementació de les diferents solucions es destina un clúster de màquines del RDI2.

1.7. Relació d'activitats

Les tasques i activitats a dur a terme en el projecte són les llistades a continuació i estan agrupades dins tres grans blocs.

1. **Planificació del projecte.**
   - 1.1. Debat de propostes.
   - 1.2. Elecció del projecte.
   - 1.3. Realització del pla de treball.
   - **1.4. Entrega PAC1.**
2. **Investigació i realització.**
   - 2.1. Cerca d'informació sobre instrumentació i tractament de dades.
   - 2.2. Cerca d'informació i teoria sobre *Hadoop*.
   - 2.3. Instal·lació i configuració *Hadoop*.
   - 2.4. Programació algorisme *MapReduce*.
   - 2.5. Cerca d'informació i teoria sobre *Spark*.
   - **2.6. Entrega parcial.**
   - 2.7. Instal·lació i configuració *Spark*.
   - 2.8. Adaptació algorisme *MapReduce* per a *Spark.*
   - 2.9. Interfície d'accés a dades.
   - 2.10. Proves fiabilitat i tolerància a errades.
   - 2.11. Comparacions i alternatives.
3. **Documentació i presentació.**.
   - 3.1. Memòria del projecte.
   - 3.2. Presentació Virtual.
   - 3.3. Autoinforme de Competències Transversals.
   - **3.4. Entrega final.**

1.8. Anàlisi de riscos

El risc principal d'un projecte d'aquest tipus es centra en els problemes temporals. S'ha de tenir cura de minimitzar l'impacte que es pugui produir en patir una situació de risc no desitjada donada per falta de temps. Per a evitar aquesta situació de risc es aconsellable realitzar un seguiment del projecte seguint la planificació inicial indicada en aquest document. A més, cal emmarcar i respectar l'abast del projecte en tot moment.

Per altre banda, la falta de disponibilitat de maquinari durant la realització del projecte i les dades que es generen poden ser, també, causes d'una situació de risc.

1.9. Abast de la proposta

El projecte es divideix en tres àmbits d'actuació bàsics que es mostren seguint el diagrama de flux del gràfic següent:

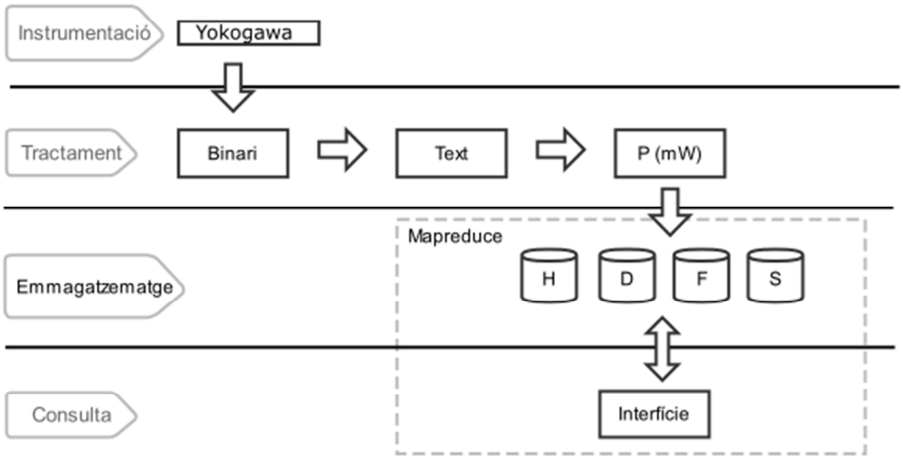


Figura 1 Diagrama de flux d'etapes del projecte.

Inicialment, cal entendre el tipus de dada a tractar i com es disposa dins els arxius que la contenen, es parla dels fitxers binaris. El tractament del text que s'extreu es fonamental per a guardar posteriorment la informació necessària. Per a la utilització posterior de les dades ens cal convertir-les de

Mitjançant l'execució de tasques *MapReduce* en una combinació formada pel parell clau i dada, d'altre manera no serà adient la utilització d'aquest tipus de computació paral·lela.

La prova de les diferents opcions d'emmagatzematge i l'elecció de la més adient forma part de la segona etapa del treball. La interfície d'accés a les dades serà l'última etapa i ha de permetre accedir a la informació necessària de potència en diferents espai de temps de manera còmode i senzilla.

1.10. Fites del projecte

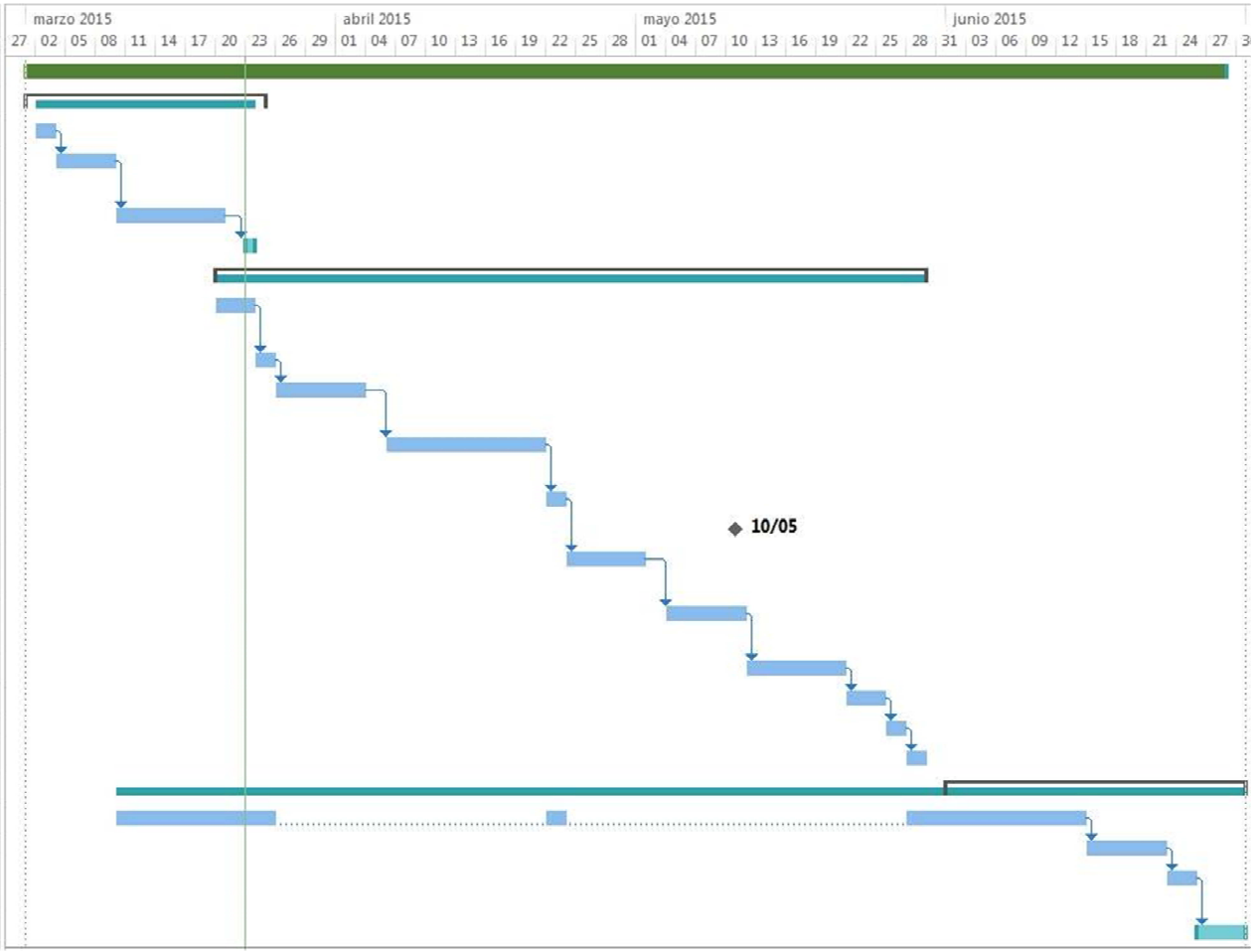
Fites	Descripció
F1	Entrega planificació del projecte (PAC1 – 23/03/2015)
F2	Estudi dels *frameworks* que s'utilitzaran.
F3	Disseny i implementació. (PAC2 10/05/2015)
F4	Documentació.
F5	Entrega i presentació final. (PAC3 28/06/2015)

Figura 2 Taula de fites del projecte.

1.11. Calendari

En aquest apartat s'estableix el calendari de referència per a l'execució del projecte. Es formen tres grans blocs, en primer per la planificació, en segon per al treball i per acabar amb la finalització de la documentació que s'anirà elaborant. L'inici del projecte correspon al dia 1 de març del 2015 i l'acabament al 28 de juny del 2015.

Moc de	Nombre de tarea	Duraci	Comienzo	Fin	Predecesora:
	TFG	87 días	dom 01/03/15	dom 28/06/15	
	Planificació del projecte	**18 días**	**dom 01/03/15**	**mar 24/03/15**	
	Debat de propostes	2 días	lun 02/03/15	mar 03/03/15	
	Selecció de projecte	4 días	mié 04/03/15	lun 09/03/15	3
	Pla de treball	9 días	mar 10/03/15	vie 20/03/15	4
	Entrega PAC1	1 día	lun 23/03/15	lun 23/03/15	5
	Investigació i realització	**51 días**	**vie 20/03/15**	**vie 29/05/15**	
	Teoria sobre Instrumentació i dades	2 días	vie 20/03/15	lun 23/03/15	
	Teoria Hadoop	2 días	mar 24/03/15	mié 25/03/15	8
	Instal·lació, configuració Hadoop	7 días	jue 26/03/15	vie 03/04/15	9
	Programació algorisme MapReduce	12 días	lun 06/04/15	mar 21/04/15	10
	Teoria Spark	2 días	mié 22/04/15	jue 23/04/15	11
	Entrega parcial	0 días	dom 10/05/15	dom 10/05/15	
	Instal·lació, configuració Spark	6 días	vie 24/04/15	vie 01/05/15	12
	Adaptació algoritme MapReduce per a Spark	6 días	lun 04/05/15	lun 11/05/15	14
	Interfície d'accés a dades	8 días	mar 12/05/15	jue 21/05/15	15
	Proves fiabilitat i tolerància	2 días	vie 22/05/15	lun 25/05/15	16
	Comparacions i alternatives	2 días	mar 26/05/15	mié 27/05/15	17
	Conclusions	2 días	jue 28/05/15	vie 29/05/15	18
	Documentació i presentació	**22 días**	**lun 01/06/15**	**mar 30/06/15**	
	Memòria del projecte	26 días	mar 10/03/15	dom 14/06/15	
	Presentació virtual	6 días	lun 15/06/15	lun 22/06/15	21
	Autoinforme de Competències tranversals	3 días	mar 23/06/15	jue 25/06/15	22
	Entrega final	3 días	vie 26/06/15	mar 30/06/15	23



1.12. Sistema resultant

El sistema resultant està format pel maquinari d'instrumentació i l'equip on s'emmagatzemen les dades. A més, calen màquines per a la implementació del clúster *Hadoop*. No es essencial el càlcul de les màquines ja que degut a les característiques del sistema es possible escalar el nombre en cas que sigui necessari. Una d'aquestes és el node principal o *namenode* que gestiona les tasques per als altres nodes anomenats *datanodes* que realitzen l'execució dels treballs. En el mateix *namenode* s'executa un servidor *Tomcat*[13] que permet l'accés web al destinatari de la informació i des d'on s'executaran les comandes en *Java* per a generar els gràfics. Per tant, en relació amb el programari, el sistema consta d'un parell d'aplicacions en Java. Una s'executa com un *Servlet*[14] al servidor web i l'altre basada en el paradigma *MapReduce* s'executa dins l'entorn *Hadoop* repartint els treballs d'extracció d'informació en els diferents nodes. És a l'inici de la funció *Map*[15] on es destria la informació útil dins els arxius binaris generats pel maquinari *Yokogawa*. Cal per tant extreure les dades dels arxius binaris mitjançant una implementació pròpia de la classe *RecordReader*[16].

1.13. Organització de la memòria

La memòria s'ha estructurat en les sis parts descrites a continuació.

1. **Introducció:** Es realitza un preàmbul del treball i es detallen aspectes com el seu context, la motivació i l'abast d'aquest a més dels recursos i la planificació temporal.
2. **Instrumentació i dades:** Es detalla la font i obtenció de les dades i l'estudi del contingut del fitxer binari.
3. **Big Data:** Introducció teòrica i informació de les diferents tecnologies.
4. **Disseny:** Explicació del sistema. El seu disseny, alternatives i decisions.
5. **Implementació:** Configuració de les diferents parts que formen el sistema.
6. **Conclusions:** Detall del conjunt de conclusions i idees extretes un cop acabat el projecte.

2. Instrumentació i dades

En la majoria dels casos, un procés depèn de variables físiques que influeixen directament en l'èxit d'aquest i on una variació d'aquestes variables pot ser la causa d'un mal funcionament en el procés. La lectura i control d'aquestes variables es un element bàsic pel manteniment i per extraure coneixement que permeti assegurar i optimitzar els processos.

En conseqüència, calen sistemes destinats a mesurar, enregistrar i controlar aquestes variables. Per a la funció esmentada el sistema d'instrumentació consta d'un dispositiu, d'un o més instruments de mesura, de les connexions entre ells i del programari i configuracions que permeten l'automatització del sistema.

En definitiva, el sistema d'instrumentació converteix una variable física en una de mesurada que s' enregistra i es fàcilment quantificada per a ser visualitzada i tractada adientment.

2.1. Equip de mesura

El pilar bàsic del sistema d'instrumentació es l'aparell de mesura. Aquest, permet mitjançant sensors, l'adquisició de la variable o variables físiques que interessa monitoritzar. Per aquest projecte s'utilitza com a referència l'oscil·loscopi *DL850* de *Yokogawa*. De totes les configuracions d'aquests equips complexos ens interessa el mètode amb el que es poden extraure les dades. En concret, aquest equip de mesura permet la connexió i l'extracció de dades mitjançant un *Usb* tipus B, una interfície *GP-IB* o mitjançant cable *Rj-45*. Tant amb *Usb* com amb *GP-IB*, la connexió directa a un ordinador permet la utilització del programari *Xviewer*[17] o *XwirePuller*[18] que facilita l'extracció i l'emmagatzematge de les mesures.

Per altre banda, una opció a tenir en compte es la connexió de l'equip de mesura mitjançant xarxa i l'emmagatzematge de les dades directament en

sistemes d'arxius remots, en aquest cas, directament en el sistema d'arxius *HDFS* del clúster.

2.2. Programari

Aquest aparell de mesura es connecta a un equip amb Windows 7, entorn en el qual s'executa el programari propietari *Xviewer* de *Yokogawa Electric Corporation*. El programari esmentat permet l'extracció de les dades dins arxius binaris en format *WDF*[19], i així, el sistema implementat recull la informació dels arxius *WDF* emmagatzemats per aquest programari per al seu tractament en el clúster.

2.3. Dades

Per a la obtenció d'informació útil cal una font de dades i un sistema que permeti la seva transformació. La font per aquest sistema es l'equip d'instrumentació i el processament d'aquestes dades es realitza directament dels arxius amb format *WDF.* De tota la informació que aquest fitxer conté ens interessa sobretot la dupla data, en format petjada de temps, i potència en *mW*.

El fitxer està estructurat en tres parts diferenciades. En primer lloc, conté una capçalera comuna per a tot l'arxiu, a la que segueix un conjunt de blocs on s'emmagatzemen les dades i finalitza en un bloc de tancament.

Per una banda, la capçalera conté 4 tipus de contingut diferent. Els 4 primers *bytes* indiquen el tipus d'arxiu mitjançant la representació d'aquest en text pla, en el cas dels arxius que s'estan tractant és *%WDF*. Els següents 4 *bytes* contenen la representació en hexadecimal del nombre total de *bytes* que formen la capçalera. Aquesta nombre total menys els primers 8 *bytes* formen la mida del bloc de dades de la capçalera. A partir d'aquest punt la resta de fitxer esta compost per blocs estructurats a partir de quatre parts; Un nom únic que identifica el bloc, la mida de les dades i una marca que indica el final del bloc. Els quatre primers *bytes* de cada bloc ens indiquen el tipus de bloc que es tracta mitjançant quatre caràcters únics. Aquests ens serviran per destriar el bloc on es llegiran les dades de la resta.

Els següents 8 *bytes* representen en hexadecimal la mida en *bytes* que ocupen les dades que formen el bloc. A continuació dels 12 primers *bytes* de cada bloc segueix la resta que acaba de formar les dades del bloc fins a un la marca de finalització que ocupa els últims 4 *bytes* i es representada amb el valor hexadecimal *0xffffffff*.

```
------------------ HEADER ------------------
File type: %WDF
Header length: 64 bytes.
--------------------------------------------
------- CHUNK: FhDR -------- Chunk length: 164 EndMark: ffffffff

------- CHUNK: HsTA -------- Chunk length: 20 EndMark: ffffffff

------- CHUNK: AcQI -------- Chunk length: 392 EndMark: ffffffff

------- CHUNK: RtMI -------- Chunk length: 140 EndMark: ffffffff

------- CHUNK: BhDR -------- Chunk length: 304 EndMark: ffffffff

------- CHUNK: thUM -------- Chunk length: 55362 EndMark: ffffffff

------- CHUNK: UnTI -------- Chunk length: 4 EndMark: ffffffff

------- CHUNK: SlTI -------- Chunk length: 16 EndMark: ffffffff

------- CHUNK: HsTI -------- Chunk length: 404 EndMark: ffffffff

------- CHUNK: HsTD -------- Chunk length: 796 EndMark: ffffffff

------- CHUNK: baLD -------- Chunk length: 1536 EndMark: ffffffff

------- CHUNK: PoTI -------- Chunk length: 1088 EndMark: ffffffff

------- CHUNK: GrMI -------- Chunk length: 132 EndMark: ffffffff

------- CHUNK: GrMW -------- Chunk length: 762568 EndMark: ffffffff

------- CHUNK: DrAW -------- Chunk length: 1089096 EndMark: ffffffff

------- CHUNK: RoFS -------- Chunk length: 2752 EndMark: ffffffff

------- CHUNK: FeND -------- Chunk length: 0 EndMark: ffffffff
```

Figura 3 Representació de la capçalera i els blocs de l'arxiu WDF

Segons la documentació del fabricant, el bloc amb la informació d'anàlisi de les dades es identificat amb *XHDR* o *DHDR* amb l'anàlisi realitzat s'ha obtingut la informació temporal a partir de l'arxiu *HDR*. Per altre banda, si s'extrau del bloc directament cal aplicar els modificadors següents:

$$Estampa\ temporal = HResolution * (Data - 1) + HOffset$$

Per altre banda, el bloc amb l'identificador *FrAW* és on s'emmagatzemen les dades de potència. Cada valor ocupa 2 *bytes* i està representat com a un número *short* amb signe. A cada element s'aplica la següent modificació:

$$Potència\ = VResolution\ *\ Valor\ binari\ +\ VOffset$$

Degut al que ocupa cada element, la mida del bloc de dades ascendeix al doble de la quantitat d'elements, fet que ha facilitat destriar la informació de potència. La capacitat d'aquest maquinari d'instrumentació permet l'extracció de valors de potència cada mil·lèsima de segon, i per tant, produint un volum elevat per a ser processat en un sol computador. Cal una solució per a tractament de grans col·leccions de dades. Tot i amb això, es realitza una implementació en codi per a extreure dades de carpetes amb arxius en moments puntuals directament a l'ordinador que captura les dades. El codi d'aquesta s'adjunta a l'Annex 1 i 2.

L'aplicació s'ha anomenat *DCW* i es escrita en *Perl* amb un llançador en format B*atch* que permet ser executada en una màquina amb una instal·lació de *Perl* i amb sistema operatiu *Windows*. Aquesta agafa els arxius *WDF* de la carpeta *DATA\IN* i els converteix en arxius de sortida a la carpeta *DATA\OUT* amb els parell marca temporal i potència de l'arxiu d'entrada.

La estampa de temps només marca 10 dígits, és a dir, en segons. Per a solucionar-ho cal agafar els últims dígits del camp *time*, extraure'n els tres primers corresponents a les mil·lèsimes de segon i adjuntar-los als anteriors i així obtenir el valor temporal desitjat.

3. *Big Data*

El terme sorgeix dins un marc d'evolució constant de les noves tecnologies i una reducció del cost en l'emmagatzematge i en la generació de les dades. Amb l'aparició de nous sistemes d'informació i la realització de noves configuracions, la producció de dades de sensors, imatges, vídeos, registres de diferents aplicacions, transaccions de diferents tipus, informació de xarxes socials, planes web entre d'altres augmenten de manera considerable.

L'interès per extreure informació d'aquestes col·leccions de dades defineix un terme que permet unificar diferents metodologies i tecnologies sota la definició de *Big Data*. Es fonamental entendre les tres dimensions en constant creixement que el formen, volum d'informació, velocitat de transmissió i la varietat del tipus de dades. Aquests tres aspectes conflueixen en un quart de resultant que es també en constant creixement, el valor. Simplificant-ho, a major quantitat d'informació, més ràpida i de diferent tipus major aportació de coneixement per a prendre decisions de valor.

Els genèrics passos a seguir a l'hora de treballar amb aquests sistemes són:

1. **Recol·lecció de Big Data**: L'organització ha d'ésser capaç de realitzar una correcta estratègia per a acumular les dades a tractar.

2. **Neteja i correlació**: Cal netejar la gran quantitat de dades d'aquells que no serveixen, el millor moment és durant la introducció de les dades. Per a correlació s'entén com el coneixement y la vinculació d'aquelles dades de diferents fonts que es refereixen al mateix objecte i s'enllacen per a ser estudiades més convenientment.

3. **Anàlisi**: És el punt clau, normalment no es tenen problemes per a recollir dades, els problemes sorgeixen a l'hora d'analitzar-les.

Es tractes d'eines que s'utilitzen per a l'anàlisi, eines de valoració predictiva, de visualització de patrons, entre altres...

4. **Conclusions:** És el punt on es vol arribar. L'anàlisi de les dades ha afegit un valor a aquella informació i ja es poden prendre decisions de valor sobre aquesta.

El sistema utilitzat en el projecte es basa en els quatre passos anteriors.

3.1. Apache Hadoop

A la practica per a treballar amb el concepte *Big Data* calen eines. L'entorn de programari idoni per aquests casos és *Apache Hadoop*. El projecte d'alt nivell d'*Apache* disponible en la versió 2.7.0 està escrit en *Java* i es desenvolupat sota llicència lliure *Apache License 2.0*.

L'entorn permet l'execució de codi escrit dins el paradigma *MapReduce* de forma distribuïda, amb la escalabilitat com a característica important ja que permet la seva execució en un únic servidor o en milers de màquines. Està inspirat en les tecnologies *Google MapReduce* i *Google File System (GFS)* i n'és d'aquesta segona d'on va sorgir el nucli principal, el seu sistema de fitxers distribuït anomenat *HDFS*, del que es parla a l'apartat 3.3 d'aquesta memòria. La configuració típica dels clústers inclouen un node principal encarregat de distribuir el treball i múltiples nodes secundaris que s'encarreguen de realitzar les tasques. Per a executar-ho cal tenir una instal·lació d'*Ssh* i el *Java Runtime Environment (JRE) 1.6* o superior. Per altre banda, l'entorn és multi plataforma.

D'es d'*Apache* es mantenen diversos projectes relacionats que permeten ampliar funcionalitats i millores com *Ambari, Avro, Cassandra, HBase, Hive*, entre d'altres. Per adonar-se de la importància que té només cal veure la quantitat d'exemples que es llisten a la plana *PoweredBy* de la vikipèdia d'*Apache*. També serveixen per veure com de diferents poden ser les configuracions i per adonar-se de la gran escalabilitat.

Per una banda, des de *Telefonica Research*, on indiquen que l'utilitzen per a la mineria de dades, la modelització d'usuaris, multimèdia i recerca de grups a internet. El clúster està format per 6 nodes amb 96 nuclis, 8 *GB* de memòria *RAM* i 2 *TB* per màquina.

I per altre banda, exemples com *LinkedIn* que utilitza múltiples *xarxes computacionals* dividides segons propòsits. Per exemple, realitzen tasques de descoberta de persones conegudes. El maquinari és en aquest cas de:

~800 màquines basades en dos processadors *Westmere HP SL 170x* amb quatre nuclis, 24 *GB* de memòria *RAM* i 12 *TB* de disc.

~1900 màquines de *SuperMicro* basades en dos processadors *Westmere X8DTT-H* de sis nuclis, 24 GB de memòria RAM i 12 TB de disc cada una.

~1400 màquines de *SuperMicro* basades en dos processadors *Sandy Bridge* de sis nuclis, 24 GB de memòria RAM i 12 TB de disc cada una.

La computació realitzada amb aquest entorn implementa el paradigma anomenat *MapReduce* que es descriu en el següent punt.

3.2. *MapReduce*

L'assoliment gradual de petites fites es realitza mitjançant el paradigma de computació paral·lela *MapReduce*. Aquest es basa en *Divide and conquer*[20] per a grans col·leccions de dades. Com indica, implica dividir el treball en processos de menor mida i tractar-los de manera distribuïda.

MapReduce consta de dues fases. Per començar, s'inicia amb la fase *Map* que duu a terme la recollida de totes les dades d'entrada, el filtratge d'aquestes i la seva classificació. Posteriorment la fase *Reduce*[21] s'encarrega d'agrupar les parts i combinar-les segons claus iguals per a realitzar les operacions que calgui . El node principal comença l'execució del treball que divideix l'entrada en diferents parts i les reparteix en els diferents nodes que formen el clúster. A

cada node es tracta la part corresponent segons el codi establert a la funció *Map*. Les dades es filtren i d'elles es conserva allò útil en forma de dupla *<clau, valor>.* Un cop filtrades i formatades s'agrupen segons clau i es tracten dins la funció *Reduce* per a generar els resultat desitjat.

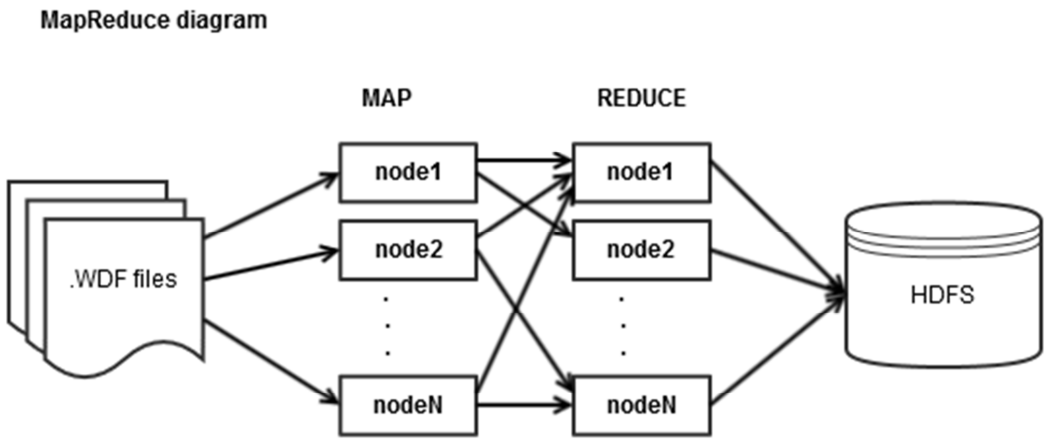


Figura 4 Diagrama *MapReduce.*

Amb l'ús de *MapReduce*, les dades resten emmagatzemades repartides pels diferents nodes que formen el clúster. El node principal revisa i envia les dades al node més proper per tal d'evitar l'enviament innecessari de dades entre nodes i la generació de tràfic de xarxa. Aquest emmagatzematge es possible gràcies al sistema de fitxers *HDFS*.

3.3. *Hadoop Distributed File System.*

El sistema de fitxers que utilitza *Hadoop* és un sistema propi anomenat *Hadoop Distributed File System* l'acrònim del qual és *HDFS*. Aquest sistema està pensat per treballar amb grans quantitats de dades de forma distribuïda, és a dir, el sistema està compost per parts localitzades en els diferents nodes que formen el clúster, el conjunt d'aquestes parts forma el sistema de fitxers. S'evita una gran quantitat de tràfic de xarxa degut a que no es mantenen les dades en un sol node i, per tant, no s'han d'enviar cap als nodes destí on es realitzaran les operacions.

Com permet replicació a les dades a nivell d'arxius és un sistema de fitxers tolerant a errades en cas de fallida d'un node. Una altre característica pròpia d'*HDFS* es la seva escalabilitat, permet l'adició de nodes a mida que en calen augmentant l'espai així l'espai del sistema de fitxers. Per altre banda, el node principal s'encarrega de mantenir el registre de quins fitxers conté el sistema i com estan disseminats entre els nodes. En *HDFS* normalment s'utilitzen blocs de mides de 64 *MB* i 128 *MB* pel fet d'estar optimitzat per a grans volums de dades, la reducció d'operacions de lectura i escriptura faran que es redueixi el temps totals de lectura i escriptura.

4. Disseny del sistema

El disseny del sistema es basa en tres etapes. En la primera, es generen les dades a l'etapa d'instrumentació on l'equip de mesura registra i guarda a l'ordinador les dades bolcant-les en els arxius binaris propis de *Yokogawa*.

El fitxer en format *WDF* es selecciona amb el codi adjunt a l'Annex 4 i es tracta amb el de l'Annex 5 per a extreure l'estampa temporal i usar-la per donar nom a l'arxiu. De seguit, s'envia al node principal per a que es carregui dins el sistema de fitxers distribuït *HDFS*. Un cop carregat ja està disponible al clúster per al seu tractament. Així doncs, agrupats els fitxers dins la carpeta d'entrada es moment de l'execució de l'aplicació *MapReduce*. Tant la càrrega com l'execució posterior es realitza mitjançant el codi de l'Annex 6.

Per altre banda, es important tenir en compte que gràcies a que el fitxer binari ocupa menys espai en disc que el fitxer de text l'enviament i posterior tractament també disminueix l'ús de la xarxa. Amb uns 1.211 *kB* aproximadament d'espai del fitxer en text i 731 *kB* d'espai de l'arxiu binari resulten per cada minut de dades un estalvi a l'enviament de 580 *kB*. 835,300 *kB* d'estalvi en transferència diaris.

El projecte en codi *Java* s'executa sota l'entorn *Hadoop* i permet llegir mitjançant la classe *WdfRecordReader*[22] , adjunta a l'Annex 9, cada un dels fitxers i enviar el flux *FSDataInputStream*[23] com a valor al node escollit per a executar la funció *Map*. En sistemes *Hadoop* amb col·leccions de fitxers de text de *GB* i *TB*, cal dividir l'arxiu en parts i enviar-les als diferents nodes. En aquest cas l'entrada del sistema està formada per molts arxius amb mides molt menors a la del bloc del sistema de fitxers, i es per això que no es parteixen en parts els fitxers d'entrada i s'executen cada un d'ells en un treball *Map* diferent. Aquest fet permet mantenir l'estructura del fitxer binari sense modificar.

Així doncs, durant la segona etapa s'executa l'algorisme, comentat a continuació, per a cada un dels arxius.

4.1. Disseny de l'algorisme segons el model *MapReduce*

En un sistema *MapReduce* típic, per a dur a terme el processament distribuït de les dades en els diferents nodes es realitzen particions als arxius d'entrada segons la mida dels blocs del sistema de fitxers *HDFS*. Aquest sistema de divisió dels fitxers d'entrada s'anomena *Split*[24], en anglès, i es essencial en els casos on els arxius tenen una mida molt superior a la del blocs del sistema de fitxers que per a *HDFS* sol ser de 64 *MB*.

La opció *dfs.blocksize* dins la configuració del node principal permet ajustar el bloc a 747861 *bytes* gairebé la mida dels arxius binaris d'entrada i que, per tant, passen a ser considerats directament cada un d'ells com una part o *Split*. D'aquesta manera l'arxiu binari es tracta tot complet i s'evita una lectura errònia. Això es degut a que al ser la mida dels fitxers d'entrada inferior al bloc del sistema de fitxers. Es tracta cada un en una instància diferent de la funció *Map*. Una alternativa a definir la mida de bloc superior al fitxer es la superposició de la funció *isSplitable()* de la classe *FileInputFormat*[25] amb un retorn del *boleà* amb valor *false*.

Per altre banda, per al disseny de l'algorisme cal definir la sortida desitjada. Una possible opció a l'hora d'agrupar les dades sorgeix a partir del caràcter únic del valor temporal i del caràcter coincident dels valors que adquireix la potència, és a dir, l'agrupació de les marques de temps per potències iguals.

Com menor es el nombre de decimals utilitzats per a definir la potència menor es la seva dispersió. Amb aquest tipus d'agrupació es divideix la sortida generada durant un minut i composta per 60.000 duples clau, valor en 60.000 estampes temporals repartides per cada potència segons els diferents valors d'aquestes. Al existir certa dispersió la quantitat de registres es elevada.

$$\begin{array}{lllll} power1 & (timestamp0, & timestamp1, & \dots & timestampN) \\ power2 & (timestamp0, & timestamp1, & \dots & timestampN) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ powerN & (timestamp0, & timestamp1, & \dots & timestampN) \end{array}$$

Figura 5 Organització per a usar *MapReduce*.

Per altre banda, hi ha la possibilitat de realitzar una agrupació per a cada valor temporal. Aprofitant que aquest és únic i consecutiu, tots els valors de potència que formen part del mateix minut s'agrupen sota el valor inicial de temps. Amb un total de 59.999 elements temporals menys a tractar per cada minut es redueix a l'hora d'emmagatzemar les dades en 84.960.000 elements temporals cada dia.

$$\text{timestamp0 (power0,} \quad \text{power1,} \quad \text{power2} \ \dots \ \text{power59999)}$$

Figura 6 Organització per a només *Map*.

En el moment de mostrar els 60.000 valors que formen cada un dels minuts només cal generar la sèrie de valors temporals a partir de l'inicial. També es important tenir en compte l'entrada que s'ha de tractar, les més típiques de *Hadoop* es realitzen amb *Text*. En canvi per aquest cas s'ha de llegir d'arxius binaris. Per tant, cal implementar la classe pròpia *WdfFileInputFormat*[26] hereva de *FileInputFormat* per a permetre l'entrada d'objectes de tipus fitxers.

Figura 7 Flux segons tipus d'entrada i sortida.

Per a l'extracció de les dades es crea la classe *WdfRecordReader* hereva de la classe *RecordReader*. La primera és un lector propi per a aquest tipus de fitxer que genera un objecte *WdfRecodReader* que permet fer una sola lectura amb tots els *bytes* que formen el fitxer, i que s'utilitzaran a la classe *Map*. El parell clau, valor són un *Text* on es desarà la marca temporal i un *FSDataInputStream* amb el flux del fitxer sencer independentment del tipus de dades. Aquest flux

llegeix els *bytes* i els converteix en caràcters *unicode*[27]. Amb l'ús d'un buffer es millora el rendiment de lectura.

Figura 8 Sortida de classe *WdfRecordReader*.

També cal tenir present que les interfícies per a un algorisme basat en *MapReduce* consten de les signatures que es mostren en la Figura 9:

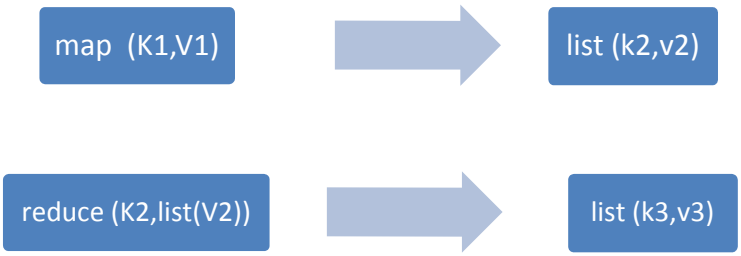


Figura 9 Interfícies típiques MapReduce.

Per la seva banda, la classe *PowerMapper* hereva de *Mapper* rep el flux d'entrada i n'extreu la informació escrivint-la en el context que rebrà la classe *Reduce*. En aquest cas, el valor consta de 60.000 registres i per tant es defineix com una *Array*, Com ha d'implementar la classe *Writable*[27] per a poder ser tractat per *Hadoop* es crea la classe *PowerArrayWritable*[28] on s'encapsularan tots els valors representats per variables de tipus *Double*.

Figura 10 Dupla de sortida del *Mapper.*

Les classes creades per a *MapReduce* han d'implementar la propietat *serializable*[29], per a ser capaces de convertir l'objecte en una seqüència de bytes i a la inversa. Aquesta propietat aporta persistència, és a dir, permet emmagatzemar les dades. En cas de no ser *serializable* durant l'execució retornà l'excepció *NotSerializableException*. Per a poder comparar entre els objectes de la mateixa classe la clau ha d'implementar la interfície *WritableComparable*[29], ja que gràcies a mètode *compareTo* es podran ordenar.

El controlador de la configuració i les execucions dels diferents treballs s'ha anomenat *PowerApplicacion*[30]. Aquest crea l'objecte *Configuration*[31] que dóna accés als diferents paràmetres dels treballs i defineix opcions com els tipus entrades i sortides de cada classe, nombre de les tasques *Reduce* i les rutes d'entrada i sortida. A més s'encarrega de gestionar la plataforma i d'inicialitzar el treball per a cada *Split*. El codi d'aquest s'adjunta a l'Annex 7. A l'Annex 10 el de la funció *Map* i a l'Annex 12 el de *Reduce*.

El resultat final es guarda en la carpeta data dins el sistema de fitxers *HDFS* amb la marca de temps com a nom de l'arxiu i els valors de potència llistats a l'interior.

Figura 11 *Passos WdfRecordReader - Mapper - HDFS*

4.2. L'alternativa *Apache Spark*

Tot i poder-se executar com un complement de *Hadoop, Spark* és un nou model de computació distribuïda que sobrepassa les deficiències de *MapReduce* sobre certs treballs iteratius i en anàlisi amb consultes, és a dir, millora el rendiment en situacions on s'executa la mateixa instrucció repetidament sobre les mateixes dades i en d'altres on cal fer consultes reiteratives a grans conjunts. També permet treballar en memòria amb una millora de la velocitat ja que evita la latència de l'accés a disc i al mateix temps manté l'escalabilitat i la tolerància a falles pròpies de *Hadoop*.

A més, permet crear dos tipus de variables útils, per una banda les anomenades *Broadcast* que són compartides entre els nodes com a només lectura i per altre, un tipus de variable anomenat acumuladors on els nodes tenen accés per afegir valors i on només el controlador té accés de lectura. Així

mateix, el punt fort d'aquesta tecnologia està en que realitza operacions amb col·leccions distribuïdes i abstractes anomenades *Resilent distributed Datasets (RDD)*[32] mitjançant funcions R*educe, Collect i Foreach*. Aquestes col·leccions es poden crear a partir d'arxius del sistema de fitxers *HDFS* de *Hadoop*. Respecte al llenguatge, està implementat per a programació d'alt nivell amb *Scala, Java* i *Python*.

En el cas del sistema del treball no compleix cap de les dues situacions esmentades, ni es processen unes mateixes dades repetidament ni es consulta reiterativament grans conjunts.

4.3. Interfície d'accés als resultats

La darrera etapa del sistema consta d'una interfície que permet l'accés a les dades ja tractades. Tot i els diferents projectes relacionats amb *Hadoop* existents es decideix implementar senzill i adaptable basat en una plana web allotjada al node principal servida mitjançant *Tomcat* que permet realitzar execucions d'un *applet* en *Java* per a extreure les dades i fer una mostra gràfica utilitzant la llibreria *JfreeChart*[33]. A continuació es mostra un diagrama de flux dels passos que formen el procés.

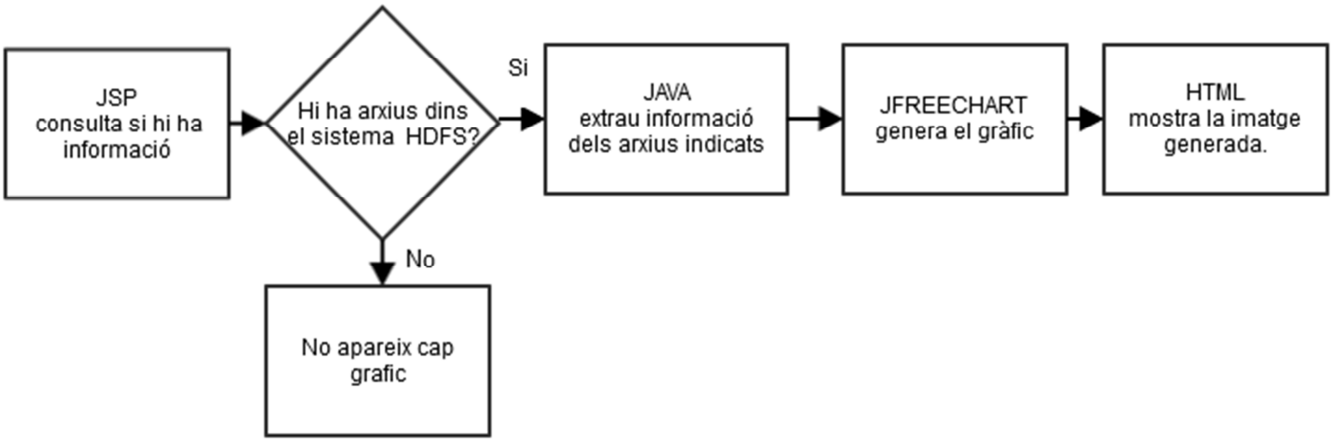


Figura 12 Diagrama dels processos de la interfície.

El codi de la interfície està format per un índex on es mostra la pantalla d'inici i es recullen les estampes temporals i un altre on es realitza la consulta de les dades del sistema *HDFS* i es mostra el gràfic. El codi d'ambdós s'adjunta a l'Annex 13.

4.3.1. *Apache Tomcat* i *Jsp.*

Apache Tomcat es una implementació de codi lliure que permet l'execució de programari en llenguatge *Java* en entorns *web*, és a dir, es un contenidor *web* que suporta *Servlets* i planes *web* escrites en *Jsp*. Aquest motor està publicat sota la llicència *Apache License 2.0*, és multi plataforma, inclou el compilador *jasper* i normalment es presenta en combinació amb el servidor web *apache*.

Per altre banda, *JavaServletPages*[34] és una tecnologia d'ajuda per als desenvolupadors de planes *web* dinàmiques basades en *Html, Xml* entre d'altres i que necessiten l'execució de codi *Java* dins d'aquestes. *Tomcat* i *Jsp* permeten realitzar consultes en *Java* dels arxius resultats i emmagatzemats al sistema de fitxers *HDFS* i mostrar la informació per navegador.

Per a la instal·lació i configuració veure el punt 5.2.

4.3.2. *JFreeChart*

El projecte *JFreeChart* és una llibreria per al llenguatge de programació *Java* que facilita la mostra d'informació en format gràfic. Aquesta llibreria inclou una *API* documentada, varietat en els tipus de gràfics i diferents formats d'imatges de sortida disponibles. Aquest projecte es distribueix sota els termes de *GNU Lesser General Public Licence (LGPL).*

Amb la recollida de la informació al servidor *web* i ja emmagatzemada en variables de la classe *series* de la llibreria *JFreeChart* permet generar un gràfic en *Png* per a mostrar dins la plana web la informació gràfica consultada.

La Figura 13 és un exemple gràfic resultant generat a partir d'una mostra correlativa d'informació ja tractada i la configuració de *JFreeChart* amb zoom al eix vertical els diferents títols.

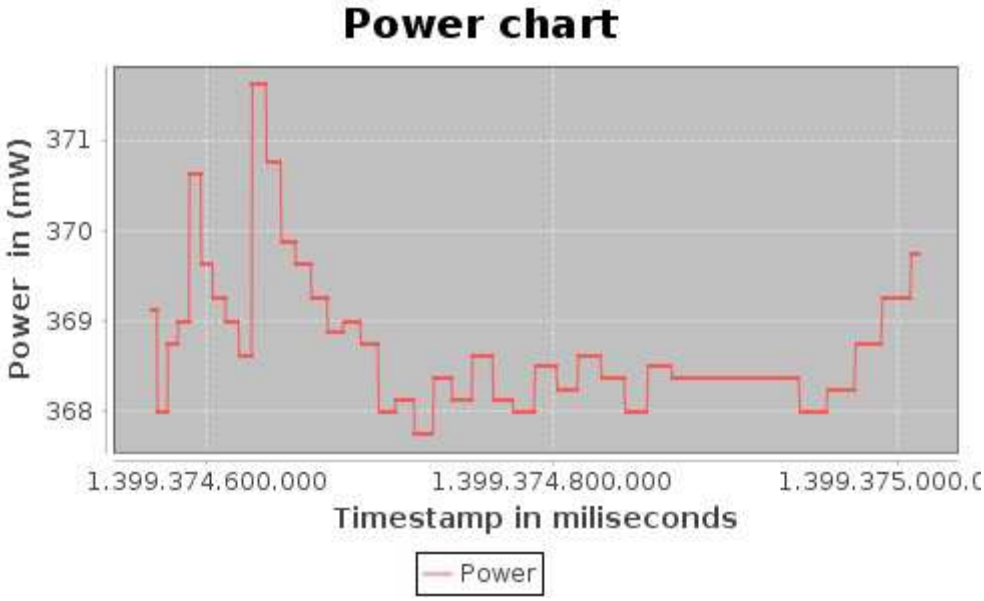


Figura 13 Exemple de gràfic resultat.

4.3.3. Interfícies alternatives

La creació d'una interfície a partir d'una plana web en *Jsp* no es la única opció a tenir en compte. Es poden emmagatzemar les dades en un base de dades no relacionals mitjançant tecnologies com *MongoDB* o *Splout SQL* que permeten la utilització del model de programació *MapReduce* per a realitzar consultes i així, millorar el rendiment prioritzant la velocitat. Bona opció en cas de gran quantitat de consultes a les dades en comptes d'accessos no simultanis. Ambdós tecnologies són un exemple de les possible aplicacions compatibles amb *Hadoop*.

Per altre banda, una opció a revisar seria la combinació de *GraphX* amb un clúster *Spark*. Aquesta interfície de programació d'aplicacions per a gràfics en paral·lel pot ser una bona implementació a l'hora d'explorar les dades i produir els gràfics resultant.

5. Implementació del sistema

Per al funcionament del sistema es necessària la instal·lació dels diferents nodes que formen el clúster i la configuració dels serveis necessaris.

5.1. Instal·lació i configuració de *Hadoop*

Per a les proves s'ha utilitzat la versió *Standalone mode* de Hadoop formada per una configuració d'un sol node, per així executar els treballs *MapReduce* en el mateix ordinador en diferents processos *Java*. En canvi, el mode clúster es l'òptim per muntar el sistema en producció. Aquest està format per diferents màquines que realitzen els treballs de manera distribuïda.

Aleshores, per a la versió individual on s'han realitzat el codi s'ha utilitzat una màquina virtual mitjançant *VirtualBox* amb *Ubuntu 14.04* i la versió *1.7* del *Jdk* de *Java* i una altre amb *Ubuntu 14.10* amb la versió 1.8 del *Jdk.* Ambdues edicions amb la versió de *Hadoop 2.6.0*, l'ultima versió estable al començament d'aquest projecte.

Ara bé, per a la instal·lació dels equips que formen part del clúster s'ha utilitzat una instal·lació Linux bàsica d'*Ubuntu 14.10* on es realitza la instal·lació i configuració pertinent de manera automàtica mitjançant l'*script* de l'Annex 3.

Posteriorment, s'afegeix a cada un dels nodes la relació *Ip* i nom de totes les màquines que formen el clúster.

```
192.168.0.2 namenode
192.168.0.3 datanode1
192.168.0.4 datanode2
```

Figura 14 */etc/hosts* dels nodes secundaris.

Al existir dos rols entre les màquines cal realitzar una configuració diferenciada segons el rol de cada un d'ells. Per una banda, cal configurar un node principal anomenat *namenode*[35] i per altre, la resta de nodes dependents d'aquest i

anomenats *datanodes*[36]. Les configuracions es realitzen en llenguatge XML en els diferents fitxers habilitats per aquestes.

Al node principal cal configurar el llistat amb la relació dels altres equips que formaren el clúster:

```
datanode1
datanode2
```

Figura 15 */usr/local/hadoop/etc/hadop/slaves* del node principal.

A més, s'ha de configurar la llista d'equips permesos i la mida del bloc del sistema de fitxers.

```
<property>
      <name>dfs.hosts</name>
      <value>names</value>
      <description>Allowed host lists</description
</property>

<property>
      <name>dfs.blocksize</name>
      <value>747861</value>
      <description>Block size in bytes</description
</property>
```

Figura 166 *hdfs-site.xml* configuració *HDFS*.

Un cop s'arrenca el servei, dins el node principal es dóna format al sistema de fitxers i es crea la ruta dins el sistema d'arxius *HDFS* a on es guarden els arxius a processar.

```
$HADOOP_HOME/bin/hadoop namenode –format

$HADOOP_HOME/bin/hadoop fs –mkdir –p input/
```

Figura 177 Format i creació de carpeta d'entrada.

Altrament, en els nodes secundaris s'indica a la configuració quin es el node principal del clúster.

```
<property>
      <name>fs.default.name</name>
      <value>hdfs://namenode:9000</value>
      <description>Namenode url:port</description
</property>
```

Figura 18 *core-site.xml* pels nodes secundaris.

A més, es configura dins l'arxiu hdfs-site.xml la llista de rutes que s'utilitzen per guardar les dades.

```
<property>
      <name>dfs.datanode.data.dir</name>
      <value>/usr/local/hadoop/data</value>
      <description>List of storage data path
      </description>
</property>
```

Figura 19 *hdfs-site.xml* amb el *path* real on es munta *HDFS*.

5.2. Instal·lació i configuració de *Tomcat*

A l'efecte d'utilitzar *Java* des de la interfície *web* es duu a terme la instal·lació i configuració bàsica de *Tomcat7* en el node principal. D'entrada, s'instal·la el paquet des del repositori, després s'exporta la configuració de variables d'entorn i tot seguit es reinicia el servei.

```
sudo apt-get install tomcat7

export CATALINA_HOME=/var/lib/tomcat7 >> ~/.bashrc

sudo service tomcat7 restart
```

Figura 20 Instal·lació *Tomcat*.

En acabar es pot accedir mitjançant l'adreça *http://localhost:8080* des del node principal o mitjançant *http://namenode:8080* des d'un equip remot.

5.3. Automatització dels processos

La idea inicial del sistema es que sigui funcional sense cap interacció. El primer per a l'automatització del sistema es la implementació de la còpia d'arxius en format *WDF* des de l'ordinador font amb *Windows 7* al node mestre de *Hadoop*. Per començar, es destria una carpeta on el programari envia els arxius *WDF* i on es desa el convertidor *WDFCon.exe*[37]. L'equipo *Windows* executa una tasca cada cert temps amb el codi que extrau les sortides en format *WVF i HDR*[38].

```
REM Extracting Header and data
FOR %%x in (*.WDF) DO \WDF2WVF\EXEWDFCon.exe %%x 1
RM *.WVF
xcopy  %cd%\*.* Z:\ /c /q /r /u /y
```

Figura 21 Execució *WDFCon.exe* i enviament al node principal.

Per això, cal compartir en el node principal la carpeta */usr/local/hadoop/input* mitjançant *Samba* i configurar aquesta ruta a l'equip *Windows* amb la lletra *Z:*\ Per altre banda, per arrencar el serveis *namenode* de *Hadoop* al iniciar el node principal es copia el codi de l'*script* de l'annex 14 a */et/init.d/* amb el nom *hadoop-namenode* i s'executa la següent comanda que permet la configuració per als diferents nivells d'arrencada *(runlevels).*

```
sudo update-rc.d /etc/init.d/hadoop-namenode start 90 2
3 4 5 .
```

Figura 22 Arrencada automàtica del node principal.

Per als nodes secundaris es configura de manera semblant canviant tant a la comanda com a l'*script namenode* per *datanode*.

```
sudo update-rc.d /etc/init.d/hadoop-datanode start 90 2
3 4 5 .
```

Figura 23 Arrencada automàtica dels nodes secundaris.

Amb les funcions de cada node assignades, i a fi de seguir amb l'execució dels treballs es descriu la comanda d'us de *Hadoop*.

```
$HADOOP_HOME/bin/hadoop  jar  $HADOOP_HOME/jar/Power.jar
input output
```

Figura 24 Comanda d'execució de codi *Java* a *Hadoop*.

La comanda del quadre anterior es realitza periòdicament mitjançant l'execució de l'Annex 6, segons l'interval marcat en la tasca següent que cal afegir a *crontab*.

```
0-59 *      * * * $HADOOP_HOME/bin/up-run-data
```

Figura 25 *Crontab.*

Depenent del funcionament inicial es por anar variant l'interval segons necessitat. Cal recordar que *MapReduce* igualment utilitza un node per a cada arxiu tant entri 1 o més d'un en cada execució. El codi que es llança des de *crontab* introdueix dins el sistema de fitxers *HDFS* els diferents arxius i inicia el seu tractament.

6. Conclusions

Com a resultat del treball presentat, s'arriba a la conclusió principal què es totalment factible realitzar un estudi en paral·lel mitjançant *Hadoop* utilitzant dades emmagatzemades directament en arxius binaris. Per a realitzar-ho és important saber la distribució del contingut d'aquests arxius, la seva codificació i adaptar la lectura de manera específica per part del sistema.

La distribució de les tasques en nodes es essencial per no carregar l'ordinador que es destina a rebre les dades de la màquina d'instrumentació. Amb l'extracció de la marca temporal directament en la fase *Map* es milloraria encara més l'estalvi de la càrrega de treball. Respecte al disseny del sistema, utilitzar *Hadoop*, una interfície en *Java* i *Tomcat* permet muntar un sistema específic per visualitzar els resultats sense utilitzar sistemes dissenyats per d'altres funcions. Quant a la lectura de les dades, es fa palesa la dificultat d'extreure la informació dels fitxers binaris i el tracte tant diferent que es té en contraposició amb els arxius de text.

Per altre banda, en relació als objectius i expectatives generades durant la planificació i que s'enumeren dins el primer apartat d'aquest projecte cal realitzar certs comentaris. Per començar, s'ha treballat amb profunditat el tractament d'arxius binaris i la relació entre les dades rebudes i la informació que s'extreu al final del procés. També cal mencionar la introducció personal al món del *Big Data*, l'augment de l'interès al respecte i la quantitat de coneixements adquirits en els diferents entorns de treball que el formen. Així mateix, s'ha assolit un cert coneixement del entorn *Hadoop* i s'han pres decisions relacionades sobre el disseny. En contra, ha faltat aprofundiment a *Apache Spark* i la falta d'adaptació del sistema i la comparació amb el presentat. Quant a la interfície, s'ha realitzar una proposta pròpia d'interfície amb tecnologies lliures a l'abast. A mode de reflexió, la majoria dels objectius establerts a la planificació s'han complert, fet que aporta satisfacció personal.

Sobre la metodologia, s'han introduït canvis respecte a la planificació degut al temps utilitzat esbrinant la localització de la marca de temps dins el binari.

Inicialment, el projecte tractava de la comparació amb *Spark* però d'aquesta només es donen quatre pinzellades i es deixa com a treball futur.

Per concloure l'apartat, s'enumeren les línies de treball que han quedat pendents. La primera d'elles i més important és l'extracció de la marca de temps directament del arxiu binari durant la fase *Map,* posició marcada a l'Annex 10. Amb aquest pas s'evita l'execució del convertidor en el equip amb Windows.

Ha quedat pendent realitzar les proves amb *Spark* per a comparar i veure quina és la millor opció, a més de l'ús de *SploutSQL* per emmagatzemar les dades i també proves de funcionament davant anomalies, parades de nodes, entrada d'arxius invàlids, i entre d'altres la revisió dels possibles problemes de seguretat del sistema. Cal per tant, una inspecció important del control d'errors.

7. Glossari

B

- ***BigData***
  [3](Anglès) A les *TIC*, és la referencia als sistemes de manipulació de grans conjunts de dades.

C

- ***Configuration***
  [31](Anglès) Classe de *MapReduce* on es configuren les diferents opcions del sistema.

D

- ***Datanode***
  [36](Anglès) Node secundari d'un clúster *Hadoop*. Aquests emmagatzemen les dades en el sistema de fitxers *HDFS* i s'encarrega d'executar els treballs *Map* i *Reduce* que li són assignats.

- ***Divide and conquer***
  [20](Anglès) Paradigma de disseny d'algoritmes basat en la divisió d'un treball per parts més simples.

- ***DL850***
  [7] Nom del model d'oscil·loscopi de l'empresa *Yokogawa* utilitzat per a l'adquisició de dades.

F

- ***Framework***
  [10] (Anglès) Entorn o marc de treball. Conjunt de recursos que permeten en grup organitzar el desenvolupament de certs projectes de programari.
- ***FSDataInputStream***
  [23] és una classe de Hadoop que permet mitjançant un *buffer* recollir un flux d'entrada de FSInputStream.
- ***FileInputFormat***
  [25] és una classe base per a tots els formats d'entrada basats en arxius.

H

- ***Hadoop***
  [4] *Framework open source* d'aplicacions distribuïdes de *The Apache Software Fundation*.
- ***HDFS***
  [9] (Acrònim anglès *Hadoop Distributed File System*) Sistema de fitxers distribuït pròpi de *Hadoop*.

I

- **Instrumentació**
  [1](Català) Sistema de control i mesura d'una o més variables que formen part d'un procés.
- **Interfície**
  [5](Català) Medi de comunicació entre un usuari i un equip que compleix cert objectiu.

J

- ***JavaServletPage***
  [12][14](Anglès) Traducció de codi *Java* i *Html* dels fitxers *Jsp* per a executar-se al servidor web. El resultat en temps d'execució es coneix com a *Servlet*.
- ***JFreeChart***
  [33] Llibreria lliure per a Java que permet als desenvolupadors generar gràfics de manera senzilla.

M

- ***MapReduce***
  [12](Anglès) Algoritme d'aplicacions distribuïdes de *The Apache Software Fundation*.
- ***Map***
  [15] Classe que format part del paradigma *MapReduce*. S'encarrega de rebre l'entrada i destriar la informació útil. Es pot codificar per afegir funcionalitats.

N

- ***Namenode***
  [35](Anglès) Node principal del clúster *Hadoop*. Node coordinador del sistema de fitxers *HDFS*, executa el controlador que distribueix els treballs entre els altres nodes.

P

- ***PowerApplication***
  [30] Controlador de les tasques *MapReduce*. Creat seguint l'*API* per al sistema del treball.
- ***PowerArrayWritable***
  [28]Classe pròpia que permet l'emmagatzemat d'un cojunt *serializable* compost per variables de tipus *Double*.

R

- ***RDD***
  [32](Acrònim anglès: Resilent distributed Datasets) Col·lecció d'elements de només lectura repartides en diferents nodes i que poden operar en paral·lel. Usat per *Spark.*

- ***RDI2***
  [6](Acrònim anglès Rutgers Discovery Informatics Institute)
- ***RecordReader***
  [16] Converteix una entrada en bytes per presentar-la a la funció *Map*.
- ***Reduce***
  [21] Funció que permet agrupar valors intermedis en una sortida segons les diferents claus. Forma part de *MapReduce*.

S

- ***Spark***
  [11] Sistema gestor de clústers de propòsit general optimitzat.
- ***Split***
  [24](Anglès) Divisió en parts d'un tot.

T

- ***TIC***
  [2](Acrònim català:Tecnologies de la informació i la comunicació) Conjunt de sistemes i tecnologies desenvolupades per a gestionar la informació.
- ***Tomcat***
  [13] Contenidor d'aplicacions desenvolupades en *Java* per a la seva execució des del *web*.

W

- ***WDF***
  [19] Format de l'arxiu binari produït pel programari.
- ***WDFCon.exe***
  [37] Programari de l'empresa *Yokogawa* que permet extraure els arxius *WVF i HDR* a partir de l'arxiu en format *WDF*.
- ***WdfFileInputFormat***
  [26] Classe pròpia hereva de *FileInputFormat* que permet el correcte tractament dels arxius binaris *WDF*.
- ***WdfRecordReader***
  [22] Classe pròpia hereva de *RecordReader*. Adaptada per una sola lectura per arxiu.
- ***WVF i HDR***
  [38] Formats d'arxius que s'extrauen amb el programa WDFCon.exe. WVF conté les dades i HDR informació de la forma en que aquestes es guarden.
- ***WdfRecordReader***
  [22] Classe pròpia hereva de *RecordReader*. Adaptada per una sola lectura per arxiu.

- ***Writable***
  [27] És una interfície de *Hadoop* implementada que permet *serializable* una entrada o a la inversa.
- ***WritableComparable***
  [29] És una interfície de Hadoop que permet la comparació d'un objecte *serializable* amb una altre. S'utilitza en la comparació de claus.

X

- ***Xviewer***
  [17] Programa de l'empresa *Yokogawa* que permet controlar els oscil·loscopis i extreure les dades en arxius .
- ***Xwire***
  [18] Programa de l'empresa *Yokogawa* que permet controlar els oscil·loscopis de la sèrie DL mitjançant una connexió *ethernet*.

Y

- ***Yokogawa***
  [8](Yokogawa Electric Corporation) Empresa fundada al 1915 de distribució d'equips d'instrumentació i control.

8. Bibliografia

- Colaboradores de Wikipedia. *Instrumentación industrial* [en línia]. Wikipedia, La enciclopedia libre, 2014 [data de consulta: Abril 2015]. Disponible a https://es.wikipedia.org/wiki/Instrumentación_industrial.

- Juan Pavón Mestras, Entrada y Salida con Java. Dep. Ingeniería del Software e Inteligencia Artificial (UCM), 2007-2008, p. 1-14

- Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, Ion Stoica. Spark: Cluster Computing with Working Sets, University of California, Berkeley 2010

- Sergi Pérez Labernia. *Big Data*. Administració de xarxes i sistemes operatius (UOC), 2013, p. 4-6

- The Apache Software Foundation. *Apache Tomcat 7* [en línia] Apache Tomcat 7.0.6.2 API, 2000-2015 [data de consulta: Juny 2015] Disponible a http://tomcat.apache.org

- The Apache Software Foundation. *Getting Started* [en línia] Apache Hadoop, 2014 [data de consulta: Abril 2015] Disponible a http://hadoop.apache.org

- The Apache Software Foundation. *Hadoop API* [en línia] Apache Hadoop Main 2.6.0 API, 2014 [data de consulta: Abril 2015] Disponible a http://hadoop.apache.org

- The Apache Software Foundation. *PoweredBy* [en línea]. Hadoop Wiki, 2015 [data de consulta: Juny 2015] Disponible a http://Wiki.apache.org/hadoop/PoweredBy

- The Perl Programming Language. *Perl.org* [en línea]. Perl.org 2002-2015 [data de consulta: Abril 2015] Disponible a https://www.perl.org/

- Yokogawa Electric Corporation. *DL850E/DL850EV ScopeCorder web* [en línia] Web de Yokogawa, 2013 [data de consulta: Abril 2015] Disponible a http://www.yokogawa.com/

- Yokogawa Electric Corporation. *Appendix 8 Using Data Files (WDF Files)*. DL850E/DL850EV ScopeCorder Features Guide IM DL850E-01EN, ed. 3a 2015, p. 266-271.

9. Annexos

Annex 1: Llançador per a l'extracció de dades per a Windows en *Batch*:

```
@echo off
echo.
echo ..........................
echo .DATA CONVERSOR FROM .WDF.
echo ..........................
echo.
REM To WDF dir
cd %CD%\DATA\IN

REM Extracting Header and data
FOR %%x in (*.WDF) DO %CD%\..\..\WDF2WVF\EXE\WDFCon.exe %%x 1

REM Storing data
FOR %%x in (*.WVF) DO MOVE %%x %CD%\..\HDR_WVF\%%x
FOR %%x in (*.HDR) DO MOVE %%x %CD%\..\HDR_WVF\%%x

REM Doing txt
FOR %%x in (*.WDF) DO perl ..\..\DCW.pl %CD% %%x

REM Return to initial dir
CD ..\..

goto:eof
```

Figura 26 Annex 1 Llançador en *Batch* per extreure dades des de Windows.

Annex 2: Extractor de dades per Windows en Perl.

```
#!/usr/bin/perl

#use strict;
#use warnings;
use Date::Parse;
use Time::Piece;

#Data directories and file names
$inHDR = $ARGV[1];
$out = $ARGV[1];
$inHDR=~ s/WDF/hdr/;
$out=~ s/WDF/txt/;

$dir = $ARGV[0];

open(IN,"<$dir/$ARGV[1]") || die("\nError openning source WDF
file.\n");
open(INHDR,"<$dir/../HDR_WVF/$inHDR") || die("\nError openning
source HDR file.\n");
open(OUT,">$dir/../OUT/$out")    ||    die("\nError    openning
destination TXT file.\n");

#Positioning Date line
$line1 =<INHDR>;
$line1 =<INHDR>;
$line1 =<INHDR>;
$line1 =<INHDR>;
$line1 =<INHDR>;

$line1 =<INHDR>;
$line1 =<INHDR>;
$line1 =<INHDR>;
$line1 =<INHDR>;
$line1 =<INHDR>;

$line1 =<INHDR>;
$line1 =<INHDR>;
$line1 =<INHDR>;
$line1 =<INHDR>;
$line1 =<INHDR>;

$line1 =<INHDR>;
$line1 =<INHDR>;
# Adquiring VResolution
      ($extra,$VResolution) = split(/\s{1,}/, $line1);

$line1 =<INHDR>;
# Adquiring VOffset
      ($extra,$VOffset) = split(/\s{1,}/, $line1);
$line1 =<INHDR>;
$line1 =<INHDR>;

$line1 =<INHDR>;
$line1 =<INHDR>;
$line1 =<INHDR>;
$line1 =<INHDR>;
$line1 =<INHDR>;
$line1 =<INHDR>;
$line1 =<INHDR>;
```

```
# Adquiring HOffset
            ($extra,$HOffset) = split(/\s{1,}/, $line1);

$line1 =<INHDR>;
$line1 =<INHDR>;
            #Extract Date as 2014/05/06
            ($extra,$date) = split(/\s{1,}/, $line1);

$line1 =<INHDR>;
            #Extract time
            ($extra,$time) = split(/\s{1,}/, $line1);

    #my $timestamp = `date +%s%N -d"$date $time"`;
    #my $ts = int $timestamp/1000000;

    #my $timestamp = ($time + 11644473600) * 10000000;

    print "$date $time <- ";

            my $timestamp= str2time($date." ".$time);
            #$timestamp = int $timestamp*1000;
    #$timestamp->epoch * 1000 + timestamp->millisecond;
   #my $timestamp = print 1000 * timestamp->epoch + timestamp-
>millisecond;

    #$timestamp = localtime($timestamp)->strftime('%Y %m %d %H
%M %S');
    print "$timestamp ";

            my $count = 0;

            #Reading header
            read (IN, $fileType, 4, 0);
      read (IN, $buffer, 4,0);
      $length = unpack('H*', $buffer);
        $length = hex($length)-8;
            read (IN, $buffer, $length, 0);

            #Reading chunks
            while(read (IN, $chunkType, 4, 0)){

read (IN, $buffer, 8, 0);
                  $lengthData = unpack('H*', $buffer);
            $lengthData = hex($lengthData);

            #Reading Data chunk
                  if($chunkType eq FrAW ){

              read (IN, $buffer, 64, 0);
              $lengthData = $lengthData - 64;

              while($lengthData > 0){
                 read (IN, $buffer, 2, 0);
              $buffer = unpack('B*', $buffer);
              $buffer = oct("0b".$buffer);

              #Value modifications
              $buffer= $buffer*$VResolution+$VOffset;

            $buffer = sprintf("%.7g", $buffer);
              $power = $buffer*3;
```

```
                $timestamp = $timestamp + $count;
                $count++;

                printf OUT $timestamp;

                printf OUT " $power\n";

                    $lengthData = $lengthData - 2;

            }

      }else{

            read (IN, $buffer, $lengthData, 0);

      }

            #End chunk
        read (IN, $buffer, 4, 0);
            $endMark = unpack('H*', $buffer);

}
```

Figura 27 Annex 2 Extractor de dades a text en en *Perl.*

Annex 3: Instal·lador amb configuració pels nodes *Hadoop.*

```
#!/bin/bash

#Update && Upgrade
sudo apt-get -y update
sudo apt-get -y upgrade

#Install Java
sudo apt-get -y install default-jdk

# Install ssh
sudo apt-get -y install ssh

#ssh key
mkdir ~/.ssh
ssh-keygen -f ~/.ssh/id_rsa -t rsa -P ""

#Copy to all cluster computers to have access
cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys

# Disable Ipv6
sudo echo
sudo echo # Disable IPV6
sudo    echo    net.ipv6.conf.all.disable_ipv6    =    1    >>
/etc/sysctl.conf
sudo   echo   net.ipv6.conf.default.disable_ipv6   =   1   >>
/etc/sysctl.conf
sudo echo net.ipv6.conf.lo.disable_ipv6 = 1 >> /etc/sysctl.conf

# Group
#sudo addgroup hadoop
#sudo adduser --ingroup hadoop
#sudo usermod -a -G hadoop hadoop
#sudo adduser hadoop sudo

#Download
cd /usr/local
sudo    wget    http://apache.rediris.es/hadoop/common/hadoop-
2.6.0/hadoop-2.6.0.tar.gz

#tar
sudo tar -xvzf hadoop-2.6.0.tar.gz

# Home link
sudo ln -s hadoop-2.6.0/ hadoop

#sudo chown -R hadoop:hadoop /usr/local/hadoop-2.6.0

# Folders
mkdir /usr/local/hadoop/input
sudo chmod 750 /usr/local/hadoop/input

# Java Home
echo  >> ~/.bashrc
echo # Variable entorno Java >> ~/.bashrc
echo export JAVA_HOME=/usr/lib/jvm/default-java >> ~/.bashrc
```

```
# Hadoop Home
echo  >> ~/.bashrc
echo # Hadoop Home >> ~/.bashrc
echo export HADOOP_HOME=/usr/local/hadoop >> ~/.bashrc


sudo shutdown -h now
```

Figura 28 Annex 3 *Script* d'instal·lació automàtica dels nodes.

Annex 4: Llançador en *Batch* d'extractor i enviament.

```
@ECHO OFF

REM @author Sergi Pérez Labernia
REM @see TFG UOC
REM @version 28/06/2015 CC BY-SA
REM Extracting initial timestamp

FOR %%x in (*.WDF) DO WDFCon.exe %%x 1
FOR %%x in (*.WVF) DO DEL %%x

REM Setting times
FOR %%x in (*.HDR) DO perl time.pl  %%x
FOR %%x in (*.WDF) DO MOVE %%x TOSEND\%%x
FOR %%x in (*.HDR) DO DEL %%x

REM Copying to namenode (revise params)
XCOPY  %CD%\TOSEND\*.* Z:\ /c /q /r /u /y
DEL /Q %CD%\TOSEND\*
EXIT
```

Figura 29 Annex4 Llançador del extractor i enviament al *datanode*.

Annex 5: Extractor d’estampa temporal i renom en *Perl*.

```
#!/usr/bin/perl

# @author Sergi Pérez Labernia
# @see TFG - UOC
# @ 28/06/2015 CC BY-SA
# Extract timestamp and rename file

#use strict;
#use warnings;
use Date::Parse;
use Time::Piece;

#file name
$IN = $ARGV[0];


open(INHDR,"$IN")   ||   die("\nError   openning   source   HDR
file.\n");

#Positioning Date line
for (my $i=0; $i < 29; $i++) {

      $line1 =<INHDR>;

}

#Extract Date as 2014/05/06
($extra,$date) = split(/\s{1,}/, $line1);

$line1 =<INHDR>;

#Extract time
($extra,$time) = split(/\s{1,}/, $line1);

#Convert to epoch
my $timestamp= str2time($date." ".$time);

#Substr miliseconds
my $milis = sprintf ($time);
$ milis = substr($milis, 9,3);

#Renaming file with miliseconds.
$IN =~ s/hdr/WDF/;
rename ($IN, $timestamp.$milis.".WDF");

close(INHDR);
```

Figura 30 Annex 5 Extracció d’estampa de l'arxiu *HDR i renom*.

Annex 6: Càrrega d'arxius a *HDFS* i execució treballs *MapReduce.*

```
#!/bin/bash
#
# Upload data to HDFS and execute Hadoop jobs

FILES=/usr/local/hadoop/input/*

for file in $FILES

do
      echo "Processing $file file...!"

      if [ $file -eq "*.WDF"]; then

            $HADOOP_HOME/bin/hadoop fs -put $file /input
            $HADOOP_HOME/bin/hadoop jar
$HADOOP_HOME/jar/Power.jar input $file

      else
            echo "$file is not an input file."

      fi
done
```

Figura 31 Annex 6 Càrrega d'arxius a *HDFS* i execució.

Annex 7: Classe *PowerApplication,* controlador en *Java* per a *MapReduce.*

```
package hadoop.power;

import java.io.IOException;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.conf.*;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

/**  <p>This  class  defines  PowerApplication.  Extends  class
configuration and set work preferences. </p>
 * @author Sergi Pérez Labernia
 * @version 28/06/2015 A
 * @see http://research.google.com/archive/mapreduce.html
 */
public class PowerApplication extends Configuration{

      /** <p>This class defines PowerApplication. Extends class
configuration and set work preferences. </p>
       * @param arg[0] input path
       * @param arg[1] output path
       */
      public static void main(String[] args) throws Exception,
IOException {

      Configuration conf = new Configuration();
      Job job = Job.getInstance(conf, "Power");

    // Map output types
    job.setMapOutputKeyClass(Text.class);
    job.setMapOutputValueClass(PowerArrayWritable.class);

    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(PowerArrayWritable.class);
    job.setMapperClass(PowerMapper.class);
    job.setReducerClass(PowerReducer.class);

    job.setInputFormatClass(WdfFileInputFormat.class);
    job.setOutputFormatClass(TextOutputFormat.class);

    //No reducer tasks
    job.setNumReduceTasks(0);

    //Paths
    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    System.exit(job.waitForCompletion(true) ? 0 : 1);
      }
}
```

Figura 32 Annex 7 Controlador *MapReduce*.

Annex 8: Classe *WdfFileInputFormat,* lectora de fitxers *WDF*.

```
package hadoop.power;

import java.io.IOException;

import org.apache.hadoop.fs.FSDataInputStream;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.InputSplit;
import org.apache.hadoop.mapreduce.TaskAttemptContext;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;

/**  <p>This  class  extends  FileInputFrmat.  Allows  to  create
WdfFileInputFormat,  WDF  file  objects  like  input  formats.
Provides getSplts from fileInputFormat. </p>
 * @author Sergi Pérez Labernia
 * @version 28/06/2015 A
 *                                                          @see
https://hadoop.apache.org/docs/current/api/org/apache/hadoop/ma
pred/FileInputFormat.html
 */
public         class         WdfFileInputFormat         extends
FileInputFormat<Text,FSDataInputStream>{

      /** <p>This method creates a new WdfRecordReader. </p>
       * @param InputSplit Split assigned to reader
       * @param TaskAttenmptContext new context
       * @return returns new WdfRecordReader
       */
    @Override
    public WdfRecordReader createRecordReader(InputSplit split,
TaskAttemptContext                context)                throws
IOException,InterruptedException {
      return new WdfRecordReader();
    }
}
```

Figura 33 Annex 8 Classe *WdfFileInputFormat*.

Annex 9: *WdfRecordReader, lectora del flux de bytes.*

```
package hadoop.power;

import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FSDataInputStream;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.InputSplit;
import org.apache.hadoop.mapreduce.RecordReader;
import org.apache.hadoop.mapreduce.TaskAttemptContext;
import org.apache.hadoop.mapreduce.lib.input.FileSplit;

/** <p>This class defines a WdfRecordReader. Extends class
RecordReader. Allows to read for one time from a input file
</p>
 * @author Sergi Pérez Labernia
 * @version 28/06/2015 A
 * @see http://research.google.com/archive /mapreduce.html*/
public class WdfRecordReader extends RecordReader<Text,
FSDataInputStream>{

    private long pos, start, end;
    private FSDataInputStream inputData;
    private Path file;
    private Text key = null;
      /** <p>This method takes assigned InputSplit and context
and prepares the Reader.</p>
       * @param genericSplit assigned Split
       * @param context assigned context
       */@Override
    public void initialize(InputSplit genericSplit,
TaskAttemptContext context) throws IOException {

      // This InputSplit is a FileInputSplit
        FileSplit FileInput = (FileSplit) genericSplit;
        Configuration job = context.getConfiguration();

        // Start ends controls when is completed
        start = FileInput.getStart();
        end = start + FileInput.getLength();

        // Retrieve file's name
        file = FileInput.getPath();

        FileSystem fs = file.getFileSystem(job);
        inputData = fs.open(FileInput.getPath());

        // Positioning for read
        this.pos = start;
    }
       /** <p>This method read a single key/value and returns
false if is read.</p>
        * @return true if exist not read data
        * @return false if data already read
        */
```

```
        @Override
        public boolean nextKeyValue() throws IOException {

      /** <p>This method read a single key/value and returns
      false if is read.</p>
             * @return true if exist not read data
             * @return false if data already read
             */
          @Override
          public boolean nextKeyValue() throws IOException {

             //Allready readed
             if (pos == end ){ return false; }

             //Finishing records.
             pos = end;

             return true;
          }
```

Figura 34 Annex 9 Classe *WdfRecordReader*.

Annex 10: Codi funció Map

```
package hadoop.power;

import java.io.IOException;
import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.Calendar;

import org.apache.hadoop.fs.FSDataInputStream;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.lib.input.FileSplit;

/**  <p>This  class  PowerMapper  extends  Mapper  class  from
MapReduce.  Takes  FSDataInputStream  and  extracts  data  to  a
PowerArrayWritable. </p>
 * @author Sergi Pérez Labernia
 * @version 28/06/2015 A
 * @see http://research.google.com/archive/mapreduce.html
 */
public     class     PowerMapper     extends     Mapper<Text,
FSDataInputStream, Text, PowerArrayWritable> {

      private  PowerArrayWritable  powerValuesWritable  =  new
PowerArrayWritable();
      // Number of power values. (Change it for another input)
      private int values = 60000;

      //Modifiers  value  (Setting  from  WDF  file  in  direct
timestamp read version)
      private double VResolution = 4.166666667E-002;
      private int VOffset = 0 ;

      /** <p>Map class. owerMapper extends Mapper class from
MapReduce.  Takes  FSDataInputStream  and  extracts  data  to  a
PowerArrayWritable. </p>
       * @see http://research.google.com/archive/mapreduce.html
       * @param key Text parameter NULL.
       * @param inputData FSDataInputStream with stream of file
bytes.
       * @param context Output context Context formed by Text
and a PowerArrayWritable with power values.
       */
      public   void   map(Text   inputKey,   FSDataInputStream
inputData,     Context     context)     throws     IOException,
InterruptedException {

            //Reading filename
            FileSplit                fileSplit                =
(FileSplit)context.getInputSplit();
            String filename = fileSplit.getPath().getName();
            filename = filename.substring(0,13);
            Text key = new Text(filename);

          byte[] type = new byte[4];
```

```
            // FileType Extraction
            inputData.read(type,0,4);

            // Reading header Data length
            int headerDataLength = inputData.readInt()-8;

        // Skipping header Data
            inputData.skip(headerDataLength);

            // ChunkType extraction (for each chunk)
            for (int chunk = 0; chunk < 17; chunk++ ){

                  inputData.read(type,0,4);
                  String chunkType = new String(type);

                  int chunkDataLength = 0;

            // Reading Data length
                  try{

                        chunkDataLength         =         (int)
inputData.readLong();

                  }catch(IOException e){

                  }

                  // Reading waveform information
                  if ("GrMW".equals(chunkType)){

                        /*  Position  where  we  suspect  timestamp
is.
                         * In direct extrat implementation, read
timestamp and
                         * setting the key. Read VResoltion and
VOffset and HResolution
                         * too.
                         * NOW SKIPPING DATA
                         */
                        inputData.skip(chunkDataLength);


            // Reading power data
                  }else if("FrAW".equals(chunkType)){

                  // Skipping start offset
                        inputData.skip(64);

                        for(int i = 0; i < values; i++ ){

                              try{

                                    double        power        =
inputData.readShort();
```

```
                                    //    Value    Modifications

                                    power                       =
(Math.floor((power*VResolution+VOffset)*1e4)/1e4)*3;

                                    // Saving into arrayWritable

      powerValuesWritable.addData(power);

                              }catch(IOException e){

                              }
                        }

       }else{

             // Skiping Data from other unused chunks
             inputData.skip(chunkDataLength);

       }
            // Skipping each chunk endmark
                  inputData.skip(4);

            }

            // Printing information
        DateFormat dateFormat = new SimpleDateFormat("yy/MM/dd
HH:mm:ss");
        Calendar cal = Calendar.getInstance();
        System.out.println(dateFormat.format(cal.getTime())+"
INFO    WdfRecordReader    Key/Value    extraction    completed
successfully.");

        //Writing output to context
        context.write(key, powerValuesWritable);
      }
}
```

Figura 35 Annex 10 Funció *Map*.

Annex 11: PowerArrayWritable

```
package hadoop.power;

import java.io.DataInput;
import java.io.DataOutput;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Arrays;

import org.apache.hadoop.io.Writable;

/**   <p>This   class   defines   PowerArrayWritable,   contains
instances    of    Double    elements.This    class    implements
Writable.</p>
 * @author Sergi Pérez Labernia
 * @version 28/06/2015 A
 *                                                          @see
https://hadoop.apache.org/docs/current/api/org/apache/hadoop/io
/Writable.html
 */
public class PowerArrayWritable implements Writable{

      private ArrayList<Double> data = new ArrayList<Double>();

      /**  <p>  PowerArrayWritable  constructor  for  an  empty
writable array.</p>
       */
      public PowerArrayWritable(){

      }

      /** <p> Constructor method for a non empty array. For a
list of doubles with data values.</p>
       * @param data ArraList of Double elements.
       */
      public PowerArrayWritable(ArrayList<Double> data){
            this.data = data;
      }

      /**
       * <p>Method than returns all data.</p>
       * @return data returns complete ArrayList
       */
      public ArrayList<Double> getData(){
            return data;
      }

      /**
       * <p>Method used for set complete ArrayList to data.<p/>
       * @param data ArrayList with input Doubles.
       */
      public void setData(ArrayList<Double> data){
            this.data = data;

      }
```

```
	/**
	 * <p> Add only one power value into array.</p>
	 * @param value to power value
	 */
	public void addData(Double value){
		this.data.add(value);

	}
	/**
	 * <p> This method returns the value inserted in index
position.</p>
	 * @param index Integer marking the position to return.
	 * @return Double from index position.
	 */
	public Double getData(int index){
		return this.data.get(index);

	}

	/** <p> Class for serialize fields of this object on
DataOutput.</p>
	 * @param out DataOuptput to serialize this object into.
	 */
	public void write(DataOutput out) throws IOException{
		int length = 0;
		if (data != null){
			length = data.size();
		}

		out.writeDouble(length);
		for(int i=0; i<length; i++){
			out.writeDouble(data.get(i));
		}
	}
	/**
	 * This method deseriablizes a serializabled object.
	 * @param in DataInput to deseriablize this object from.
	 */
	public void readFields(DataInput in) throws IOException{
		Double length = in.readDouble();

		data = new ArrayList<Double>();

		for(int i=0; i <length;i++){
			data.add(i, in.readDouble());
		}
	}

	/**
	 * Method han overrides ArrayWritable toString.
	 *  @return  string  returns  String  with  all  values
represented as String.
	 */
	@Override
	public String toString()
		String string = Arrays.toString(data.toArray());
		return string;
	}
}
```

Figura 36 Annex 11 Classe *PowerArrayWritable.*

Annex 12: Codi funció *Reduce*

```
package hadoop.power;

import java.io.IOException;

import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

/**  <p>Reduce  Class.  Extends  Reducer  class  from  MapReduce.
Takes Text and
 * PowerArrayWritable to generate Text output. Not used in the
system because
 *      there      aren't      duplicated      keys,      setted
job.setNumReduceTasks(0); in driver.
 * For other calcs in other systems active and do it here </p>
 * @author Sergi Pérez Labernia
 * @version 28/06/2015 A
 * @see http://research.google.com/archive/mapreduce.html
 */
public     class     PowerReducer     extends     Reducer<Text,
PowerArrayWritable, Text, Text> {

      // Number of power values. (Change it for another input)
      private int values = 60000;
      private StringBuffer text;

      /**   <p>Reduce   Class.   Extends   Reducer   class   from
MapReduce. Taker power array to write in output file.</p>
       * @see http://research.google.com/archive/mapreduce.html
       * @param key Text with timestamp.
       *   @param   powerArray   PowerArrayWritable   with   power
values.
       * @param context Output context to write in file.
       */
      public    void    reduce(Text    key,    PowerArrayWritable
powerArray, Context context)
      throws IOException, InterruptedException {

            // ARRAY TO TEXT
            Double value;

            for (int i = 0; values < 60000; i++ ){

                  value = powerArray.getData(i);
                  text.append(value);

                  if(i != 59000){   text.append("\n");

                  }
            }

            Text powerValues = new Text(text.toString());

       context.write(key, powerValues);

    }
 }
```

Figura 37 Annex 12 Funció *Reduce.*

Annex 13: Interfície Web. Índex i mostra de resultats.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta       http-equiv="Content-Type"       content="text/html;
charset=UTF-8">
<title>Power Information by web </title>
</head>
<body>
      <table>
     <tr>
            <td>
          <h1> Power graphics generator</h1>
         </td>
     </tr>
      </table>

    <p>This page show charts about Yokogawa DL850 power:
<br>
      <form method="POST" action="./result.jsp">

      Initial   time:   <input   type="text"   name="timestamp0"
size="13" maxlength="13"><br>
      Final   time:   <input   type="text"   name="timestamp1"
size="13" maxlength="13"><br>
      <h5>*13 digits time stamp</h5>
      <br>
      <input type="submit" value="Submit">

      </form>

</body>
</body>
</html>
```

Figura 38 Annex 13 *index.jsp* página inicial i formulari entrada

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta       http-equiv="Content-Type"       content="text/html;
charset=UTF-8">
<title>Power Information by web </title>
</head>
<body>
      <%@ page import="java.io.*" %>
      <%@ page import="java.net.*" %>
      <%@ page import="java.util.*" %>
      <%@ page import="org.apache.hadoop.conf.*" %>
      <%@ page import="org.apache.hadoop.fs.*"%>
      <%@ page import="org.jfree.chart.*"%>
      <%@ page import="org.jfree.data.xy.*"%>
      <%@ page import="org.jfree.chart.plot.*"%>
      <%

      // Receiving timestamps
      String timestamp0 = request.getParameter("timestamp0");
      String timestamp1 = request.getParameter("timestamp1");
      // Parsing to Double
      Double t0 = Double.parseDouble(timestamp0);
      Double t1 = Double.parseDouble(timestamp1);
      Double seachTime = t0;
      Double range = t1 - t0;
      Double power = null;

      if ( range.intValue() > 0 ){
            // Create data serie for chart
            XYSeries series = new XYSeries("Power");
            Configuration conf = new Configuration();
            FileSystem    fileSystem    =    FileSystem.get(new
URI("hdfs://namenode:9000"), conf);
            Path path = null;

            int flag;
            for (int i = 0; i<60000;i++){

                  // Timestamp seach in 60.000 ms before.
                  //path = new Path("/"+t0+".WDF");
                  path = new Path(t0-i+".WDF");
                  flag = i;

                  if (fileSystem.exists(path) == true){

                        i=60000;
                  }

            path = new Path(t0-flag+".WDF/part-r-00000");
            if (range <= 600000){

                  //Reading from 1 file
                  FSDataInputStream     fsDataInputStream     =
fileSystem.open(path);

                  // TO IMPLEMENT READING CORRECTLY
```

```
                    BufferedReader data = new BufferedReader(new
InputStreamReader(fsDataInputStream));
                    String string;

                    int index = 0;
                    while ((string = data.readLine()) != null) {
                          power = new Double(string);
                          series.add(t0+index,power);
                          i++;
                    }
                    fsDataInputStream.close();

              }else{
                    //Reading from more than 1 file
                    int nFiles = (int)(range / 60000)+1;

                    for(int j=0; j<nFiles;i++){
                          path = new Path(t0+j+".WDF");
                          FSDataInputStream  fsDataInputStream  =
fileSystem.open(path);
                          // TO IMPLEMENT READING CORRECTLY
                          BufferedReader      data      =      new
BufferedReader(new InputStreamReader(fsDataInputStream));
                          String string;
                          int k = 0;
                          while  ((string  =  data.readLine())  !=
null) {
                          power = new Double(string);
                          series.add(t0+k,power);
                          k++;
                    }

                    fsDataInputStream.close();
                    }
              }

              fileSystem.close();

              // Add the series to data set
              XYSeriesCollection       dataset       =       new
XYSeriesCollection();
              dataset.addSeries(series);

              // Generate the graph
              JFreeChart chart = ChartFactory.createXYLineChart(

"Power chart",                      // Title
"Time in miliseconds",        // x-axis Label
"Power  in (mW)",             // y-axis Label
dataset,                            // Dataset
PlotOrientation.VERTICAL,     // Plot Orientation
true,                               // Show Legend
true,                               // Use tooltips
false                               // Configure chart to
              );

              try {
                    ChartUtilities.saveChartAsJPEG(new
File("chart.jpg"), chart, 800, 300);

              } catch (IOException e) {
```

```
                    System.err.println("Problem  occurred  creating
chart.");

            }
      }

      } else{

            %>
            <br>
            <p> Final time is older than initial.
            <br>
            <%
      }

      %>
            <img src="./chart.jpg">

            <form method="POST" action="./index.jsp">
            <input type="submit" value="Return">
            </form>

</body>
</html>
```

Figura 39 Annex 13 *resultat.jsp* Mostra gràfica resultant.

Annex 14: *Script* d’arrencada node principal.

```
#! /bin/sh
# /etc/init.d/hadoop-namenode
#

#Cambiar namenode per l’arrencada dels nodes secundaris.
case "$1" in
  start)
    echo "Starting Hadoop "
    $HADOOP_HOME/bin/hdfs namenode
    ;;
  stop)
    echo "Stopping Hadoop"
    $HADOOP_HOME/sbin/stop-all.sh
    ;;
  *)
    echo "Usage: /etc/init.d/hadoop-namenode {start|stop}"
    exit 1
    ;;
esac
```

Figura 40 Annex 14 Script arrencada node principal.