

# Apache Spark instrumentation using custom PIN Tool

sparkanalyzer

José Manuel García Sánchez

# Outline

- Presentation
- Apache Spark modifications
- Pintool development: pinSpark
- Evaluation: Spark cluster over Amazon AWS
  - Deployment, execution and instrument workloads
- Future works & conclusion

# Highlights

- Sparkanalyzer was developed as a degree project at Open University of Catalonia (UOC)
- This project is the last step in my Grade of Computer Engineering studies
- Sparkanalyzer belongs to computer architecture area at UOC

# Final Degree Project

- Sparkanalyzer was proposed by Francesc Guim (UOC) as final degree project for my studies
- Final degree project in UOC is a subject with 12 credits
- Sparkanalyzer was developed between september to january, 2016.

# Personal Goals

- ◆ Apply knowledge obtained along my grade studies to a complex IT project.
- ◆ Pass my last subject to earn 12 credits so I can graduate

# Presentation(I)

- Sparkanalyzer is a project to instrument Apache Spark workloads using PIN custom tool
- To use Sparkanalyzer, we need a custom apache spark deployment and permissions to modify libraries/source code (we need to wrap our pin tool inside the Spark code)
- Pintool records values of execution on Slaves nodes on spark.

## Presentation(II)

- Instrumentation runs on Slave nodes. We tool only instruments Jobs on the Spark cluster submitted over standard submit procedures(spark-submit.sh)
- PIN output is stored on MySQL database for central management and on local filesystem on each slave node
- PIN Tool configuration is done with a conf file, /etc/pinSpark/pinSpark.conf

# Apache Spark Modifications

- Tested over Apache Spark 1.5.1
- We need to compile and modify one file of Spark Source code. We have identified where the jobs are executed on Slaves, so our modification puts pin execution before the jobs is launched on the worker to instrument.
- `core/src/main/scala/org/apache/spark/launcher/WorkerCommandBuilder.scala`



# Apache Spark Modifications(II)

- Our custom code reads configuration values from `/etc/pinSpark/pinSpark.conf`
- After modify and compile Apache Spark, we can copy and use Apache Spark Distribution modified to instrument.
- It's necessary to run our Spark distribution on all nodes that conforms the cluster. Also `pinSpark.conf` must exists on each node

# Apache Spark Modifications(III)

- Instrumentation is only on each job that is executed on Slave nodes.
- Apache Spark modification is independent of PIN Tool. By config values on `/etc/pinSpark/pinSpark.conf` we can modify execution and parse another tool or script before job execution

# pinSpark

- Developed using PIN Tool framework
- Reads configuration values from `/etc/pinSpark/pinSpark.conf`
- Instrument execution of
  - Memory read/write, barrier, instructions, etc.
  - Syscalls: Open, Close, Read, Write, Send, Recv

## pinSpark (II)

- Stores information on MySQL database and local filesystem file on Slave node
- Simple development. The tool is very simple: Follow execution and analyze defined metrics.
- Easy to expand with custom metrics
- Tool independent of Spark. We can execute standalone

# pinSpark (III)

- PIN distribution must exist on each slave node. Instrumentation is executed locally on jobs.
- What is really launched on Spark slave?
- INFO ExecutorRunner: Launch command: `"/servers/pin/pin.sh" "-t" "/servers/pin/source/tools/pinSpark/obj-intel64/pinSpark.so" "-o" "/tmp/pinSpark_principal_RL1Zz.log" "--" "/usr/lib/jvm/java-8-oracle/jre/bin/java" "-cp" "/servers/production/spark-1.5.0-bin-pin-Spark/sbin/./conf:/servers/production/spark-1.5.0-bin-pin-Spark/lib/spark-assembly-1.5.0-hadoop2.2.0.jar" "-Xms2048M" "-Xmx2048M" "-Dspark.driver.port=52307" "org.apache.spark.executor.CoarseGrainedExecutorBackend" "--driver-url" "akka.tcp://sparkDriver@10.0.0.108:52307/user/CoarseGrainedScheduler" "--executor-id" "0" "--hostname" "10.0.0.108" "--cores" "4" "--app-id" "app-20151209175456-0000" "--worker-url" "akka.tcp://sparkWorker@10.0.0.108:36427/user/Worker"`

# Evaluation: Amazon EMC

- All the development and testing performed on ECS instances
- Compilation of Apache Spark need a large machine configuration. We choose c4.xlarge instance type for development/master node.
- Slave nodes can use any instance type. For our tests we take t2.medium

# Evaluation: Amazon EMC

Launch Instance Connect Actions

Filter by tags and attributes or search by keyword 1 to 7

<input type="checkbox"/>	Name	Instance ID	Instance Type	Availability Zone	Instance State	Status Checks	Alarm Status	Pub
<input type="checkbox"/>	PRINCIPAL	i-174b4bb6	c4.large	eu-west-1b	running	2/2 checks passed	None	ec2-
<input checked="" type="checkbox"/>	NODO2	i-ada5d226	t2.medium	eu-west-1b	pending	Initializing	None	ec2-
<input checked="" type="checkbox"/>	NODO1	i-d995d060	t2.medium	eu-west-1b	pending	Initializing	None	ec2-
<input type="checkbox"/>	NODO3	i-f0d3957b	t2.medium	eu-west-1b	stopped		None	ec2-
<input type="checkbox"/>	NODO4	i-fdd39576	t2.medium	eu-west-1b	stopped		None	ec2-
<input type="checkbox"/>	NODO5	i-fed39575	t2.medium	eu-west-1b	stopped		None	ec2-
<input type="checkbox"/>	NODO6	i-ffd39574	t2.medium	eu-west-1b	stopped		None	ec2-

## Evaluation: Amazon EMC (II)

- After download PIN, Apache Spark and compile and deploy all software, we can use Amazon AMI to generate a template for machine deployment
- Slaves nodes are deployed on EC2 using our custom AMI with all software already prepared
- We need to modify `pinSpark.conf` on each slave node



## Evaluation: Amazon EMC (III)


- We store all software on /servers directory
- Starting of master process (only on principal node)
  - `root@principal:/servers/production/spark-production/sbin# ./start-master.sh`
- Of course, under spark-production we deploy our custom and modified Apache Spark distribution

# Evaluation: Amazon EMC (IV)

- Starting of master process (only on principal node)
  - `root@principal:/servers/production/s`  
`park-production/sbin# ./start-`  
`slave.sh spark://principal:7077`
- On slave nodes we must start slave pointing to master node. We deploy a hosts file with all the machines IP for easy management, but it's not necessary

# Evaluation: Amazon EMC (V)

- We can access spark management web on <http://principal:8080>

 **Spark Master at spark://principal:7077**

URL: spark://principal:7077  
REST URL: spark://principal:6066 (*cluster mode*)  
Alive Workers: 2  
Cores in use: 4 Total, 0 Used  
Memory in use: 5.7 GB Total, 0.0 B Used  
Applications: 0 Running, 0 Completed  
Drivers: 0 Running, 0 Completed  
Status: ALIVE

## Workers

Worker Id	Address	State	Cores	Memory
<a href="#">worker-20151228132417-10.0.0.197-36610</a>	10.0.0.197:36610	ALIVE	2 (0 Used)	2.9 GB (0.0 B Used)
<a href="#">worker-20151228132447-10.0.0.146-52017</a>	10.0.0.146:52017	ALIVE	2 (0 Used)	2.9 GB (0.0 B Used)

## Running Applications

Application ID	Name	Cores	Memory per Node	Submitted Time	User	State	Duration
----------------	------	-------	-----------------	----------------	------	-------	----------

## Completed Applications

Application ID	Name	Cores	Memory per Node	Submitted Time	User	State	Duration
----------------	------	-------	-----------------	----------------	------	-------	----------

# Deploying and testing

- To submit jobs to the cluster we can use some tools. Our custom Spark binary distribution needs that you send packages with spark-submit script. This is located on bin directory.
- Spark-submit uses multiple switches to configure jobs deployed to the cluster

# Deploying and testing (II)

- This is an example of job
  - `./spark-submit --class org.apache.spark.examples.JavaWordCount --master spark://principal:7077 --executor-memory 2G --total-executor-cores 2 /servers/production/spark-production/examples/target/spark-examples_2.10-1.5.0.jar /servers/data/cantar.txt`
- We run WordCount from Spark examples, using 2 core and 2GB of Ram per slave node. Also we pass the file to run wordcount

# Deploying and testing (III)

- Execution stores output on MySQL database and also on /tmp (random file name) on each slave node. Local log file also stores debug output
- You can modify instrumentation using pinSpark.conf file (deactivate some instrumentation metrics, for example).
- pinSpark.conf also permit configuration of open/read/write syscall related to a directory (for example, instrument from /data or from complete root /)

# Deploying and testing (IV)

- As instrumentation puts a huge overhead on execution, output to log file and database is performed each a fixed number of instructions (you can modify on pin tool source code)
- This output is a snapshot of instrumentation until this moment. Values are accumulative, you can use this outputs to see how the instrumentation and execution goes. Of course, latest entry is the final value for the instrumentation
- If you run multiple workers on different slaves, information on database appears identified with custom field showing node

# Future works & conclusion

- Sparkanalyzer can be improved with (my principal topics)
  - Modify source to instrument syscalls with dynamic configuration based on config file
  - Implement a solution to study output execution with data store on database
  - Modify code for better performance
  - Create a installation script to automatize download, compilation and preparation of environment



# Future works & conclusion (II)

- We have developed a patch to instrument Spark jobs on Apache Spark
- Instrumentation only happens on worker job, so we instrument the exact job submitted to the Spark cluster, not the infrastructure software or services
- Also, we have developed an example tool that use our procedure to gather metering data from spark execution
- Source code can be downloaded from
  - <https://bitbucket.org/fguim/sparkanalyzer>