Scaling Evolutionary Programming with use of Apache Spark

Włodzimierz Funika[1,2] and Paweł Koperek[1]
[1] AGH University of Science and Technology, Faculty of Computer Science, Electronics and Telecommunications, Department of Computer Science, al. Mickiewicza 30, 30-059 Kraków, Poland
[2] ACC CYFRONET AGH, ul. Nawojki 11, 30-950 Kraków, Poland

Research motivation: mitigate hubert[3] scalability problems.
- **Support for large data sets** – limited only by available computing resources.
- Decoupling evaluation from other stages of algorithm – **refactoring towards microservice architecture**.
- Enabling usage of **computing clusters**.

Spark features
- **Clear programming model** using RDD abstraction.
- Straightforward **deployment in Mesos,** YARN[4].
- **Processing resilence**: recomputation of failed steps.

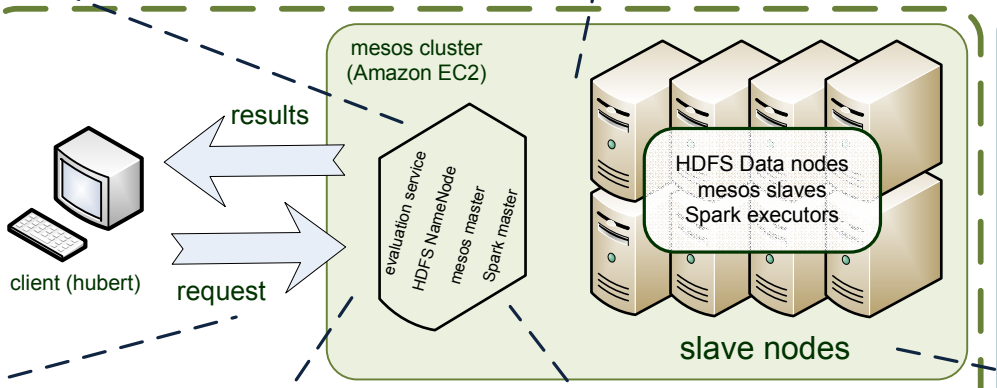
Why Spark? **Hubert** vs **Spark** assesment

Hubert	Spark
• Uses algorithm which can reuse intermediate processing results.	• Automatic in-memory caching of data sets.
• Written in Java.	• Based on Java VM platform.
• Able to utilize only a single processor.	• Cluster size up to hundreds of nodes.
• Input size limited to single machine.	• Integration with HDFS[4].
• Monolithic architecture.	• Encourages implementing services.
• Aiming to process streams of data.	• Streaming extensions available.

1 Client computes the individuals for evaluations. For each individual a separate request is created. The request contains:
- matematical expression (the individual)
- data set location
- which algorithm use for numerical differentiation

2 Evaluation service triggers creating a new Spark[1] cluster through mesos[2]. Mesos master downloads and installs Spark executors on all slave nodes. Evaluation service submits new job to newly created Spark master.

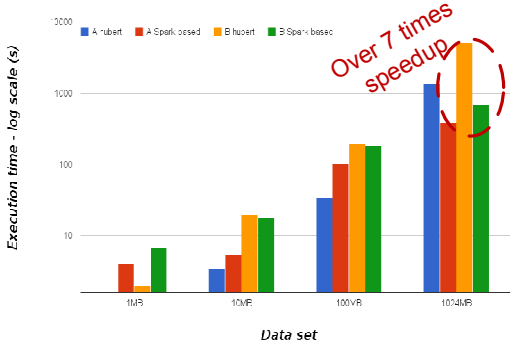
3 Spark master splits the job and sends it in form of serialized Java code to slaves. Mesos slave nodes process Spark jobs with use of Spark executors.



4 Input data is read from HDFS or local filesystems (in this case each node needs to contain the same files). Intermediate data (RDD partitions) is stored in memory to prevent subsequent *FS reads.

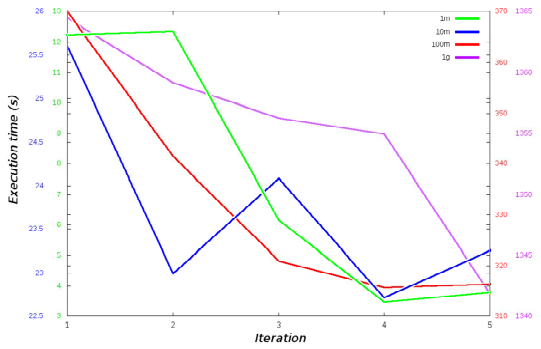
5 Spark master is notified about progress of tasks. In case a node fails or data of required RDDs is not cached anymore – recomputation of missing information is scheduled.

6 Evaluation service sends back the evaluation result. References to RDDs related to finished computations are cached in service.



Sample functions used for evaluation
A) sin(x+y)
B) (x-y+cos(x)-4.906+5.8+x-y)/(cos(4.56575)+cos(x)+sin(x)*x/y)

Evaluation results
- Best effects achieved for big datasets with more complicated computations – 7,19 speedup for 1024MB.
- For smaller amount of data overhead of framework prevails gains from parallel processing.
- Subsequent iterations are processed faster due to Spark caching mechisms.
- Spark demonstrates capacity for further improvement by using larger number of nodes, manual cache tuning, pre-computing numerical differentiation results.



Automatic caching data in memory speeds up subsequent evaluation iterations.

Source code available at: https://github.com/pkoperek/nibbler

References

1. Zaharia M., et al.: Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing. NSDI 2012. April 2012
2. Hindman B., et. Al, Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center, NSDI'11, p. 295-308
3. Funika W., Koperek P., Genetic Programming in Automatic Discovery of Relationships in Computer System Monitoring Data, In Proc. PPAM 2013, Part I, LNCS 8384, pp. 371–380, Springer, 2014
4. Apache Hadoop project: http://hadoop.apache.org/, accessed: 2014-09-11

Acknowledgements

This research is partly supported by the European Union within the EU ICT-269978 VPH-Share Project.