Finding the Big Data Sweet Spot: Towards Automatically Recommending Configurations for Hadoop Clusters on Docker Containers

Rui Zhang, Min Li* and Dean Hildebrand
IBM Research - Almaden *IBM T.J. Watson Research Center

Introduction

Aim:

---automatically configuring Hadoop workloads for container-driven clouds

Contributions:

---design a lightweight algorithm based on customized k-nearest neighbor to efficiently recommend Hadoop and container configurations prior to job execution

---identify key parameters for YARN MapReduce performance on Docker, and leverage the Sahara framework to enforce recommended configurations.

---estimate the performance gain from our approach using early experiments that consider Hadoop configurations only

System Architecture

---consider automatic configuration as an offline recommendation problem

---design a lightweight custom k-nearest neighbor (KNN) heuristic that leverages a simple intuition
--- borrowing configuration knowledge from “similar” past jobs whose configuration has delivered good performance

---denote a set of past jobs {J1,...,JN} where each job Ja is associated with with three vectors

System Architecture --- First Vector

TABLE I: Job feature vector

Feature	*Values*
Job type	{iterative, interactive, real-time, batch}
Data size	{small, medium, large}
Data type	{text, image, database}
Sensitive resource	{cpu, memory, network, i/o}
Resource load at job submission	{high, medium, low}

A job feature vector $\vec{F}a$ =< fa1, . . . , fam > containing features that are descriptive of the job

System Architecture --- Second Vector

TABLE II: Key YARN container parameters

Parameter names	*Meaning*
yarn.nodemanager.resource.memory-mb	Total memory available for all containers
yarn.scheduler.minimum-allocation-mb	Minimum memory limit per container
yarn.scheduler.maximum-allocation-mb	Maximum memory limit per container
mapreduce.map.memory.mb	Memory per mapper container
mapreduce.reduce.memory.mb	Memory per reducer container
yarn.nodemanager.vmem-pmem-ratio	Container physical vs. virtual memory ratio
mapreduce.map.cpu.vcores	Virtual cores per map container
mapreduce.reduce.cpu.vcores	Virtual cores per reduce container

TABLE III: Key Docker-specific container parameters

Parameters	*Meaning*
–m	Container memory limit
–c	CPU shares (relative weight)
—privileged	Give extended privileges to this container
–device	Run devices inside the container without the –privileged flag
–lxc-conf	lxc options including cgroup resource shares between containers
–s	storage driver: one of aufs, devicemapper, btrfs and overlay
– storage-opt:	storage driver options

A job configuration vector $\vec{C}a$ =< Ca1 , . . . , Can > consisting of analytics framework configuration parameters or cloud platform parameters

System Architecture --- Third Vector

A job performance vector $\vec{P}a =< Pa1 , . . . , Pao >$, comprising of metrics that characterize job performance (e.g. execution time, throughput, utilization)

System Architecture --- Methodology

---J_x denote a new incoming job
---$\vec{F}_x$ becomes known upon submission
---Goal: determine the job configuration vector $\vec{C}_x$ for the new job such that $\vec{P}_x$ is desirable.

---solving two sub problems

The first problem

---identify the k-nearest neighbors for the new job to form a group Gx = {Jx1,...,Jxk} that have k past jobs most similar to Jx in terms of their job feature vectors

---Solve

---define similarity

$$S(J_a, J_b) = \sqrt{\sum_{i=1}^{m} s(f_{ai}, f_{bi})^2} \qquad s(f_{ai}, f_{bi}) = \begin{cases} 1 & if \quad f_{ai} = f_{bi} \\ 0 & otherwise \end{cases}$$

---m is the size of the job feature vector, fai and fbi are the i-th element of the feature vector for jobs Ja and Jb

---per element similarity function returns 1 if the two corresponding elements of job feature vectors fai and fbi are the same

The Second Problem

---rank the performance vectors associated with each job in Gx and return the configuration vectors corresponding to the top k' (k' ≤ k) performance vectors that meet a performance threshold

---Slove

---use a weighted sum formulation to convert each performance vector into a single score

$$P(J_a) = \sum_{i=1}^{o} \beta_i \times p'_{ai}$$

---βi is a normalized weight factor

---For metrics such as response time, whose value corresponds negatively to performance, we set P'ai = 1/Pai; otherwise we simply have P'ai = Pai

---Sahara is the OpenStack component that aims to enable users to automatically provision and manage Hadoop clusters in OpenStack cloud environments

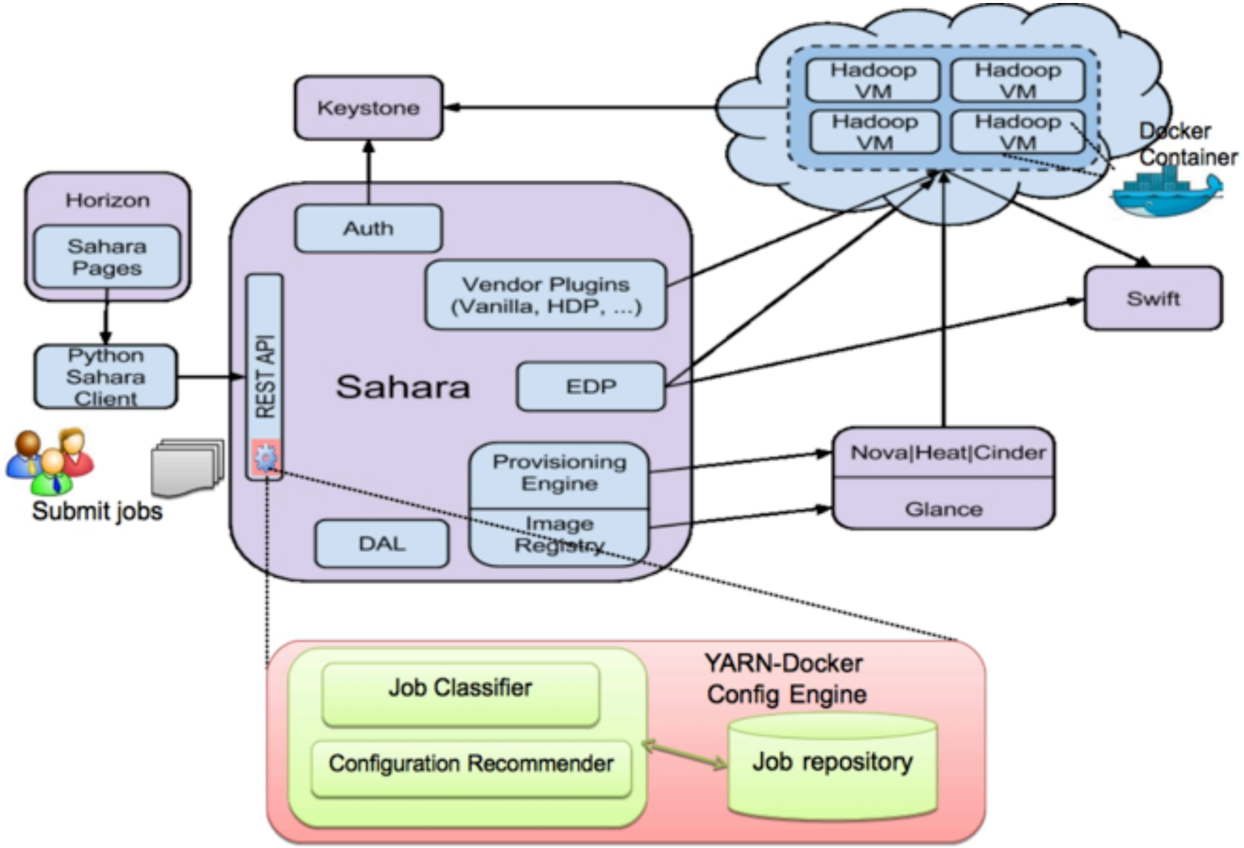


Fig. 1: Overall architecture with a view of Sahara integration

---Sahara's RESTful job APIs would allow to capture past job features, configurations and performance and are stored in the job repository

---When a new job arrives, it is first assigned a neighbor group by the classification module(the first problem) and subsequently recommended one or a few configurations (the second problem)

---The final configuration can be enforced using a Sahara transient cluster, a custom temporary cluster launched specially for the duration of one single job with a specific configuration

---three application from HiBench

Benchmark	Input Size	Shuffle Size	Output Size
Bayes	1.2 GB	47 GB	37 GB
K-means	8 GB	30 KB	4 KB
PageRank	12.8 GB	27 GB	6.4 GB

---Bayes is a CPU, memory and shuffle intensive application
---K-means is a CPU intensive application
---PageRank is a CPU and shuffle intensive web search application
---The input data sets are generated using the HiBench generator

---run every benchmark 3 times and get the average execution time for each
---comparing configurations recommended with the default configurations shipped with YARN.

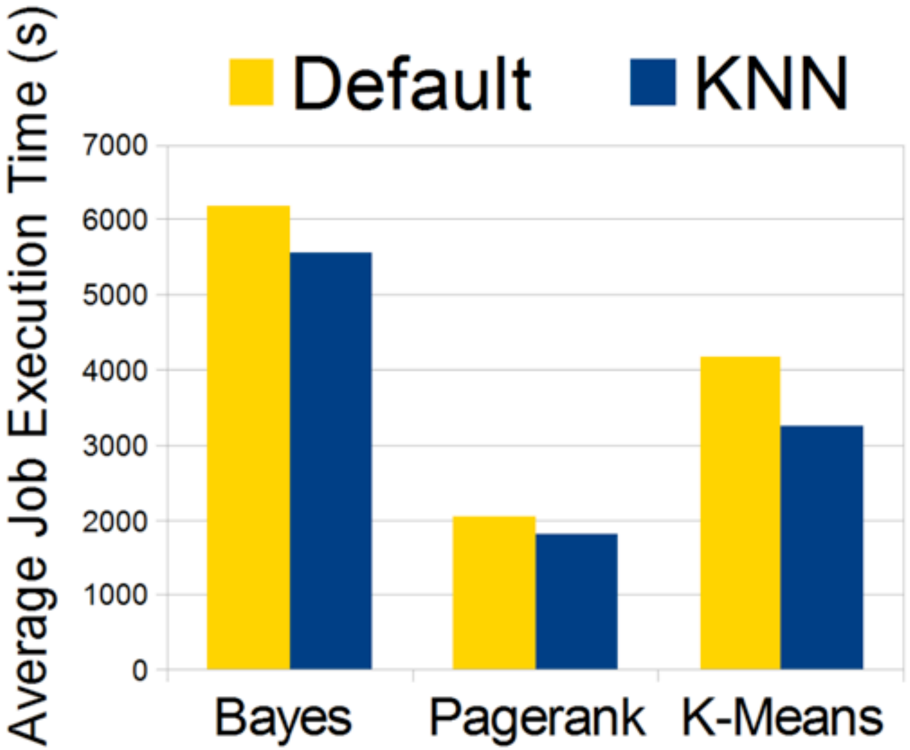
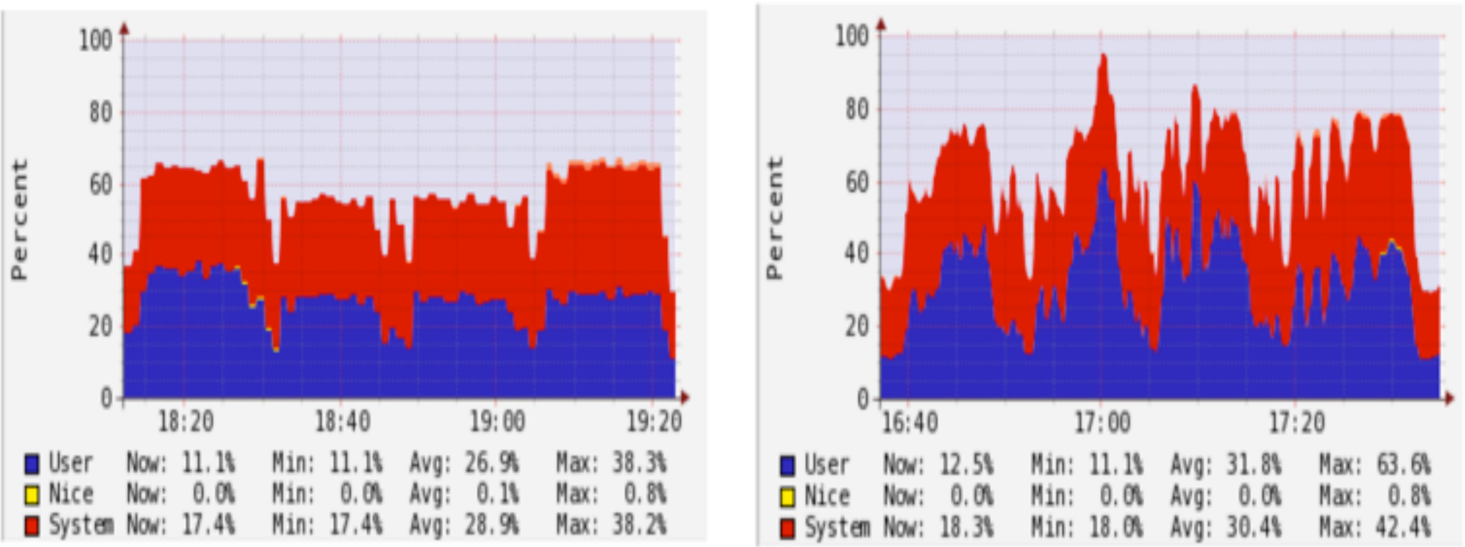


Fig. 2: Execution time: our recommended configuration vs. default.

---reduces job execution time by 11%, 13%, and 28% for Bayes, PageRank and K-means respectively



(a) Default. (b) KNN Heuristic.

Fig. 3: CPU Utilization of K-means.

---how the good configurations recommended by KNN improve resource utilization, in turn leading to performance gain

---average CPU utilization of K-means is improved by 5.9%

---The CPU utilization of K-means has four peaks corresponding to the centroid computation and three iteration of the clustering

---Similar trends can be observed for memory utilization