Web-Scale *k*-means++ Clustering on PowerPS

Zhanhao (Jasper) Liu
The Chinese University of Hong Kong
zhliu6@cse.cuhk.edu.hk

ABSTRACT

PowerPS is a general and scalable Parameter Server[1] based system for distributed machine learning. PowerPS provides more flexible control over computing resources with a novel multi-stage design than existing systems. In this research, we developed an efficient, scalable and distributed implementation of *k*-means++ Clustering algorithm on PowerPS.

KEYWORDS

Large Scale Machine Learning, Parameter Server, Distributed System, Distributed Optimization

1 INTRODUCTION

The scale of machine learning problems today is increasing in terms of both data volume and model dimensionality, and distributed methods have been widely employed for large scale machine learning. The Parameter Server (PS) framework has been widely adopted in existing machine learning systems to scale distributed machine learning nowadays.[5] To address the serious problems in the design and implementation of existing PS-based systems, which have limited the flexibility and generality of the PS abstraction, Husky team[2] develop a new design of the PS architecture and present a new system called PowerPS. PowerPS provides more flexible control over computing resources with a novel multi-stage design.

k-means algorithm is a most popular clustering method and was identified as one of the top 10 algorithms in data mining.[8] However, the traditional *k*-means algorithm encounters certain performance bottlenecks when it comes to the huge data and model size. We present 3 improvements to the popular *k*-means clustering algorithm to address the extreme requirements for latency, scalability, and sparsity encountered in user-facing web applications using PowerPS. First, we use Scalable *k*-means++[2], a parallel version of the *k*-means++ initialization algorithm and empirically demonstrate its practical effectiveness, to initialize the centers of clusters rather than choosing randomly from the dataset. Second, we adopt the use of mini-batch optimization for*k*-means clustering. This reduces computation cost by orders of magnitude compared to the classic batch algorithm while yielding significantly better solutions than online stochastic gradient descent. [7] Last, by making use of the novel multi-stage design of PowerPS, we divided the training task of the *k*-means algorithm into stages, so that the task can be assigned to run in different locations with different amounts of computing recourses in different stages to maximize the use of cluster resources.

[1] A framework in which both data and workloads are distributed over worker nodes, while the server nodes maintain globally shared parameters, represented as dense or sparse vectors and matrices.
[2] http://www.husky-project.com/

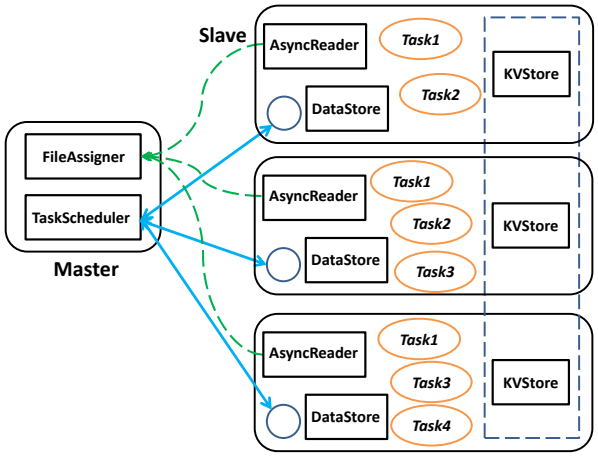

Figure 1: PowerPS System Architecture

2 BACKGROUND

2.1 PowerPS

PowerPS is organized in a master-slave architecture, as shown in 1. PS separates the working units into *workers* and *servers*, where parallel workers update models stored in the servers. Both data and workloads are distributed over slave machines, while the server nodes maintain globally shared parameters, represented as dense or sparse vectors and matrices. The parameters (key-value pairs) are stored in KV-store and support asynchronous *pull/push*.

The relaxed consistency of PowerPS further hides synchronization cost and latency. PowerPS allow the algorithm designer to balance algorithmic convergence rate and system efficiency by offering three kinds of mechanisms for model sync-up, including BSP, ASP, SSP, in a unified manner. The best trade-off depends on data, algorithm, and hardware.

2.2 *k*-means Clustering

Algorithm 1 *k*-means++ (*k*) Initialization

1: $C \leftarrow$ sample a point uniformly at random from X
2: **while** $|C| < k$ **do**
3: Sample x $\in X$with probability $\frac{d^2(x,C)}{\phi_X(C)}$
4: $C \leftarrow C \cup \{x\}$
5: **end while**

The traditional *k*-means clustering method randomly chooses K observations from the data set and uses these as the initial means, which is very fast and intuitive. However, this method often performs poorly in terms of convergence speed, especially when all

the randomly chosen data points actually belong to the same cluster. To address this problem, Arthur and Vassilvitskii proposed *k*-means++[1], an approximation algorithm for the NP-hard *k*-means problem.

2.2.1 k-means++. The main idea of *k*-means++ is to choose the centers one by one in a controlled manner, where the current set of chosen centers will stochastically *bias* the choice of the next center, see Algorithm 1. The advantage of this approach is to avoid merging clusters together like in *k*-means and to obtain an approximately optimal results on the synthetic datasets.

However, *k*-means++'s inherent sequential nature limits its applicability when it comes to web-scale data: one must make k-1 passes through the data to find all the k initial centers, which may be very time-consuming when the dataset is of great volume.

2.2.2 Scalable k-means++ (a.k.a k-means||). To reduce the number of passes needed to obtain initial centers in *k*-means++ algorithm, Bahmani proposed a Scalable *k*-means++[2] algorithm to obtain a nearly optimal solution after a logarithmic number of passes, and then show that in practice a constant number of passes suffices. see Algorithm 2.

Algorithm 2 Scalable *k*-means++ (k, l) Initialization

$C \leftarrow$ sample a point uniformly at random from X
$\psi \leftarrow \phi_X(C)$
for $O(\log \psi)$ **do**
  $C' \leftarrow$ sample each point $x \in X$ independently with probability $p_x = \frac{l \cdot d^2(x, C)}{\phi_X(C)}$
  $C \leftarrow C \cup \{C'\}$
end for

2.2.3 Mini-batch k-means. While the above *k*-means++ and*k*-means|| algorithm are mostly focused on the pre-initialization phases of *k*-means, Sculley proposed the use of mini-batch optimization for post-initialization phases of *k*-means clustering to reduce computation cost by orders of magnitude compared to the classic batch algorithm while yielding significantly better solutions than online stochastic gradient descent. [7] See Algorithm 3.

3 IMPLEMENTATION

3.1 Parameter

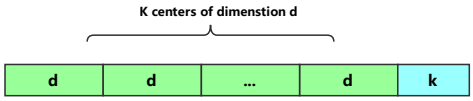

Figure 2: KV-store of Web-Scale *k*-means++ Clustering

In PowerPS framework, the parameter of a machine learning task is stored in the KV-store shared by all workers. The KV-store of PowerPS offers normal *pull*/*push* API to allow a task to access the KV-store like in other parameter server system. Besides, PowerPS also offers chunk-based *pull*/*push* API to support natural and efficient parameter accessing. All these operations are asynchronous and non-blocking. In our implementation of the Web-Scale *k*-means++ Clustering, we use the features of each center as the parameter of the whole algorithm. Also, in the mini-batch Algorithm 3, we need to maintain a vector of size k to keep track of the number of data in each cluster. We use one chunk to store the features of a center and the total number of parameter is $k \cdot d + k$, where d is the number of features in each data point. See Figure 2 for the content in the KV-store.

Algorithm 3 Mini-batch *k*-Means.

Given: k, mini-batch size b, iterations t, data set X
Initialize each $\mathbf{c} \in C$ with an $\mathbf{x}$ picked randomly from X
$\mathbf{v} \leftarrow 0$
for $i = 1$ to t **do**
  $M \leftarrow b$ examples picked randomly from X
  for $\mathbf{x} \in M$ **do**
    $d[\mathbf{x}] \leftarrow f(C, \mathbf{x})$ ▷ Cache the center nearest to x
  end for
  for $x \in M$ **do**
    $\mathbf{c} \leftarrow d[\mathbf{x}]$
    $\mathbf{v}[\mathbf{c}] \leftarrow \mathbf{v}[\mathbf{c}] + 1$
    $\eta \leftarrow \frac{1}{\mathbf{v}[\mathbf{c}]}$
    $\mathbf{c} \leftarrow (1 - \eta)\mathbf{c} + \eta\mathbf{x}$
  end for
end for

Algorithm 4 Web-Scale *k*-means++ Clustering.

Given: k, mini-batch size b, iterations t, dataset X, $\mathbf{v} \leftarrow 0$
Initialize each $\mathbf{c} \in C$ using Scalable*k*-means++ algorithm
KV-Worker $r = 1, \ldots, m$:
for $i = 1$ to t **do**
  Pull initial cluster centers C and $\mathbf{v}$ from KV-Servers
  $M \leftarrow b$ examples picked randomly from X
  for $\mathbf{x} \in M$ **do**
    $\mathbf{c} \leftarrow f(C, \mathbf{x})$ ▷ Cache the center nearest to x
    $\mathbf{v}[\mathbf{c}] \leftarrow \mathbf{v}[\mathbf{c}] + 1$
    $\eta \leftarrow \frac{1}{\mathbf{v}[\mathbf{c}]}$ ▷ Update learning rate
    $\Delta w_r \stackrel{+}{=} -\eta(\mathbf{c} - \mathbf{x})$
  end for
  push Δw_r^i to KV-servers
end for
KV-Server:
Receive initial cluster centers (w^0) from KV-Worker
for $i = 1$ to t **do**
  Send w^{i-1} to each KV-Worker
  Receive Δw^i from KV-Worker and update w^i
end for

Our implementation of the Web-Scale *k*-means++ Clustering is divided into three tasks: *load_task* (load data from HDFS), *init_task*

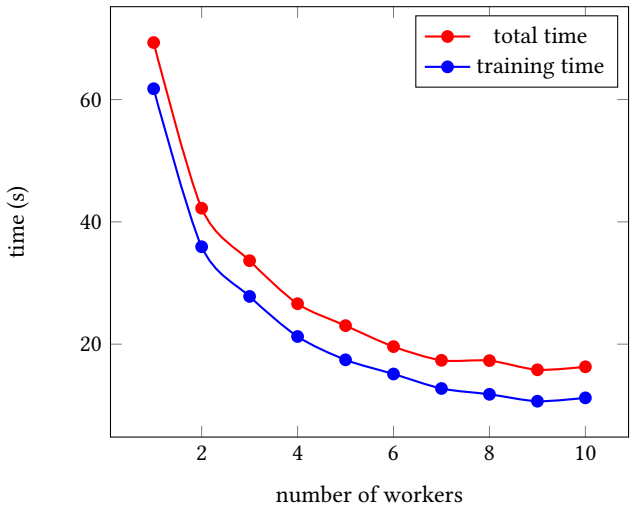

Figure 3: Scalability of Web-Scale *k*-means++ Clustering

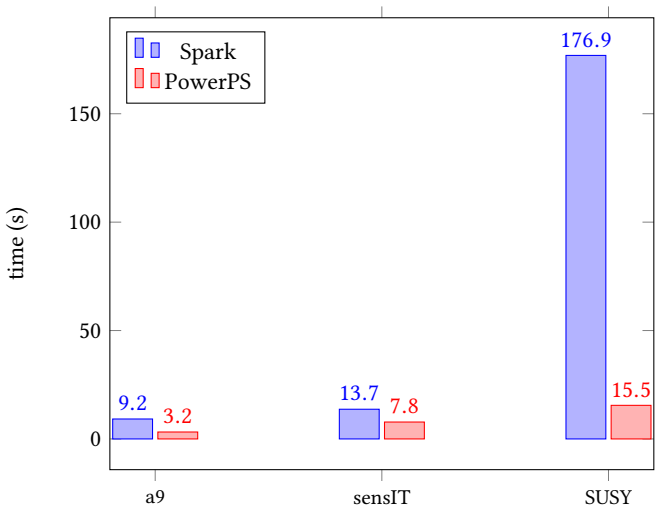

Figure 4: Convergence time of*k*-means

(initialize the k centers) and *train_task* (conduct mini-batch updating of the centers).

In data loading task, the task scheduler of PowerPS issue load_data() function for each load_worker, each of the worker loads its corresponding part of data into its local storage. In the initialization task, we offer 3 kinds of initialization methods, "random", " extitk-means++" or "*k*-means||". In the training task, each worker gets their own set of data from the datastore and pull the parameters from KV-store, and conduct a mini-batch training on its data. After the training process, each worker pushes their own set of sub-gradient to the KV-server. KV-server update all the parameters stored in KV-store according to the consistency model (BSP, ASP, SSP). See Algorithm 4 for the whole structure of our Web-Scale *k*-means++ Clustering.

4 EXPERIMENTS

We conducted several experiments on different datasets to evaluate the performance of Web-Scale*k*-means++ Clustering on PowerPS.

4.1 Scalability

The scalability of a distributed algorithm can be roughly measured by the linear relationship between the number of workers and the running time for the same dataset as shown in Figure —●—. [3] [4]

As indicated by the figure, training time and total time both decrease nearly proportional to the increment of the number of workers.

4.2 Convergence speed

Another important performance index of an distributed algorithm is its convergence speed. To evaluate the performance of our Web-Scale*k*-means++ Clustering, we conducted several experiments to use Web-Scale *k*-means++ Clustering on PowerPS and the *k*-means clustering from *Spark*[9] to cluster the same datasets and evaluate their convergence time. See Figure ▮▮ for the experiment results.

Comparing the performance of Web-Scale*k*-means++ Clustering on PowerPS with the *k*-means algorithms on Spark, a conclusion can be drawn that the former outperform Spark significantly in terms of Convergence time. See Table 1 for some information about the three datasets used in the above experiment.

Table 1: Dataset information

Name	# of classes	# of data	# of features
a9[6]	2	32561	123
SenseIT	3	78823	100
SUSY[3]	10	5000000	18

5 CONCLUSION

In this project, we presented a distributed Web-Scale *k*-means++ clustering using parameter server and adopted the multi-stage feature of PowerPS to accelerate the computation and make the most of the computing resources. In terms of scalability and convergence speed, this implementation outperforms the state of art MLlib on Spark platform.

6 ACKNOWLEDGEMENT

The author would like to express his special thanks to Tatiana Jin, Yidi Wu, Tommy Tu, Yuzhen Huang and others in Husky Team for their kind guidance and generous assistance. He also likes to thank Prof. James Cheng for giving him such a great opportunity to explore about distributed system and machine learning.

REFERENCES

[1] David Arthur and Sergei Vassilvitskii. 2007. K-means++: The Advantages of Careful Seeding. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '07)*. Society for Industrial and Applied Mathematics,

[3] (The dataset we use in this experiment is SensIT Vehicle (combined)[4], which contains 78823 data, each has 100 features and belong to one of the 3 clusters.)
[4] The time is measured by executing 10000 iterations.

Philadelphia, PA, USA, 1027–1035. http://dl.acm.org/citation.cfm?id=1283383.1283494

[2] Bahman Bahmani, Benjamin Moseley, Andrea Vattani, Ravi Kumar, and Sergei Vassilvitskii. 2012. Scalable K-means++. *Proc. VLDB Endow.* 5, 7 (March 2012), 622–633. https://doi.org/10.14778/2180912.2180915

[3] P. Baldi, P. Sadowski, and D. Whiteson. 2014. Searching for exotic particles in high-energy physics with deep learning. *Nature Communications* 5, Article 4308 (July 2014), 4308 pages. https://doi.org/10.1038/ncomms5308 arXiv:hep-ph/1402.4735

[4] Marco F. Duarte and Yu Hen Hu. 2004. Vehicle Classification in Distributed Sensor Networks. *J. Parallel Distrib. Comput.* 64, 7 (July 2004), 826–838. https://doi.org/10.1016/j.jpdc.2004.03.020

[5] Mu Li, David G. Andersen, Jun Woo Park, Alexander J. Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J. Shekita, and Bor-Yiing Su. 2014. Scaling Distributed Machine Learning with the Parameter Server. In *Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation (OSDI'14)*. USENIX Association, Berkeley, CA, USA, 583–598. http://dl.acm.org/citation.cfm?id=2685048.2685095

[6] John Platt. 1998. Fast Training of Support Vector Machines Using Sequential Minimal Optimization, In Advances in Kernel Methods - Support Vector Learning. https://www.microsoft.com/en-us/research/publication/fast-training-of-support-vector-machines-using-sequential-minimal-optimization/

[7] D. Sculley. 2010. Web-scale K-means Clustering. In *Proceedings of the 19th International Conference on World Wide Web (WWW '10)*. ACM, New York, NY, USA, 1177–1178. https://doi.org/10.1145/1772690.1772862

[8] Xindong Wu, Vipin Kumar, J. Ross Quinlan, Joydeep Ghosh, Qiang Yang, Hiroshi Motoda, Geoffrey J. McLachlan, Angus Ng, Bing Liu, Philip S. Yu, Zhi-Hua Zhou, Michael Steinbach, David J. Hand, and Dan Steinberg. 2008. Top 10 algorithms in data mining. *Knowledge and Information Systems* 14, 1 (01 Jan 2008), 1–37. https://doi.org/10.1007/s10115-007-0114-2

[9] Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, and Ion Stoica. 2010. Spark: Cluster Computing with Working Sets. In *Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing (HotCloud'10)*. USENIX Association, Berkeley, CA, USA, 10–10. http://dl.acm.org/citation.cfm?id=1863103.1863113