# Scaling Distributed Machine Learning With The Parameter Server

Wang Jian

2020 年 11 月 10 日

## 目录

- OSDI 2014

# 1 Abstract

We propose a parameter server framework for distributed machine learning problems.

# 2 1 Introduction

Sharing imposes three challenges:(带宽、同步、容错)

- Accessing the parameters requires an enormous amount of network bandwidth.
- Many machine learning algorithms are sequential. The resulting barriers hurt performance when the cost of synchronization and machine latency is high.

- At scale, fault tolerance is critical. Learning tasks are often performed in a cloud environment where machines can be unreliable and jobs can be preempted.

| ≈ #machine × time | # of jobs | failure rate |
|---|---|---|
| 100 hours | 13,187 | 7.8% |
| 1,000 hours | 1,366 | 13.7% |
| 10,000 hours | 77 | 24.7% |

Table 1: Statistics of machine learning jobs for a three month period in a data center.

对第三点的解释。Here, task failure is mostly due to being preempted or losing machines without necessary fault tolerance mechanisms.

## 2.1 1.1 Contributions

该文提出第三代的参数服务器。It confers two advantages to developers:

- enables application-specific code to remain concise
- it provides a robust, versatile, and high-performance implementation capable of handling a diverse array of algorithms from sparse logistic regression to topic models and distributed sketching.

  Our parameter server provides five key features:

  - Efficient communication
  - Flexible consistency models
  - Elastic Scalability
  - Fault Tolerance and Durability
  - Ease of Use

## 2.2 1.2 Engineering Challenges

reading and updating parameters shared between different worker nodes is ubiquitous. parameter server 保存部分参数，worker node 需要这些参数的子集。两个挑战：

- Communication parameter: key-value pairs, 值通常很小，每次跟新操作开销很大，采用这种方式低效。解决办法：参数通常表示为向量或张量等，
- Fault tolerance failover 故障转移，self-repair 能为动态扩展提供支持

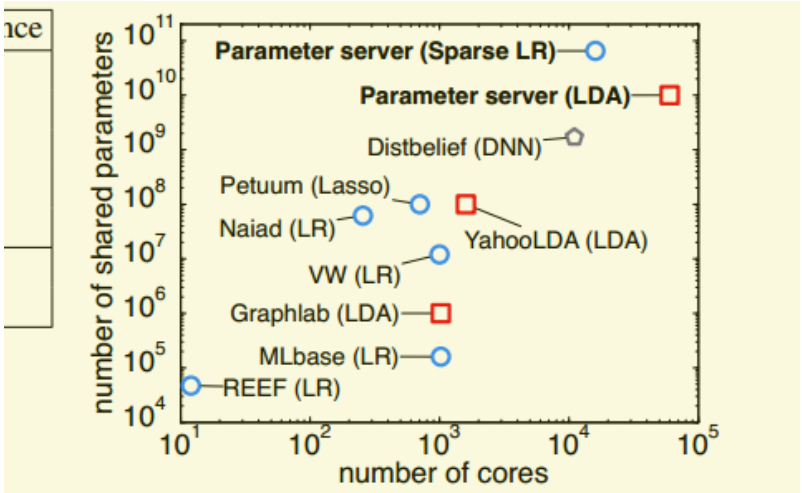


Figure 1: Comparison of the public largest machine learning experiments each system performed. Problems are color-coded as follows: Blue circles — sparse logistic regression; red squares — latent variable graphical models; grey pentagons — deep networks.

| | Shared Data | Consistency | Fault Tolerance |
|---|---|---|---|
| Graphlab [34] | graph | eventual | checkpoint |
| Petuum [12] | hash table | delay bound | none |
| REEF [10] | array | BSP | checkpoint |
| Naiad [37] | (key,value) | multiple | checkpoint |
| Mlbase [29] | table | BSP | RDD |
| Parameter Server | (sparse) vector/matrix | various | continuous |

Table 2: Attributes of distributed data analysis systems.

简而言之，我们的工作比公开的其他工作要好。

## 2.3 1.3 Related Work

发展历程：

- The first generation of such parameter servers: memcached distributed(key, value) store
- The second generation: Distbelief
- The third generation: Ours

其他：

- GraphLab 缺乏弹性的可扩展性
- Piccolo 使用参数服务器的策略进行多台机器的状态共享和聚合，他因此实现了我们系统的大部分功能，但缺少优化策略（message compression, replication, and variable consistency models expressed via dependency graphs）

# 3 2 Machine Learning

## 3.1 2.1 Goals

目标函数 (objective function), 如何快速训练大量的数据是我们的关注的重点。

## 3.2 2.2 Risk Minimization

rish – prediction error, overfit, 正则化, loss, 一些机器学习的概念等等。损失函数和正则化对本文不重要。

我们简单描述为 distributed subgradient descent: 次梯度方面只是简单地推广到损失函数和不需要连续可微的正则化器，如在 w = 0 的 |w|

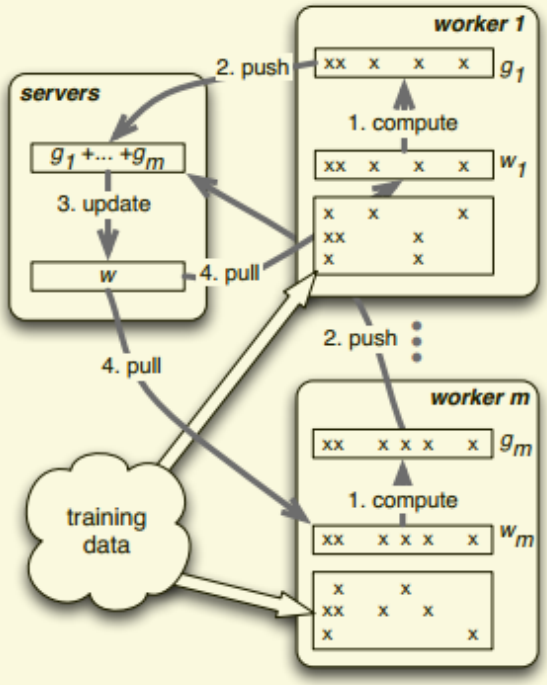


Figure 2: Steps required in performing distributed subgradient descent, as described e.g. in [46]. Each worker only caches the working set of $w$ rather than all parameters.

每个 worker 只训练自己的数据，得到 subgradient，然后聚集跟新 w。

**Algorithm 1** Distributed Subgradient Descent

**Task Scheduler:**

issue LoadData() to all workers
**for** iteration $t = 0, \ldots, T$ **do**
    issue WORKERITERATE($t$) to all workers.
**end for**

**Worker $r = 1, \ldots, m$:**

**function** LOADDATA()
    load a part of training data $\{y_{i_k}, x_{i_k}\}_{k=1}^{n_r}$
    pull the working set $w_r^{(0)}$ from servers
**end function**
**function** WORKERITERATE($t$)
    gradient $g_r^{(t)} \leftarrow \sum_{k=1}^{n_r} \partial \ell(x_{i_k}, y_{i_k}, w_r^{(t)})$
    push $g_r^{(t)}$ to servers
    pull $w_r^{(t+1)}$ from servers
**end function**

**Servers:**

**function** SERVERITERATE($t$)
    aggregate $g^{(t)} \leftarrow \sum_{r=1}^{m} g_r^{(t)}$
    $w^{(t+1)} \leftarrow w^{(t)} - \eta \left( g^{(t)} + \partial \Omega(w^{(t)}) \right)$
**end function**

时间开销最大的是计算 subgradient，但是计算被分配到多台机器。

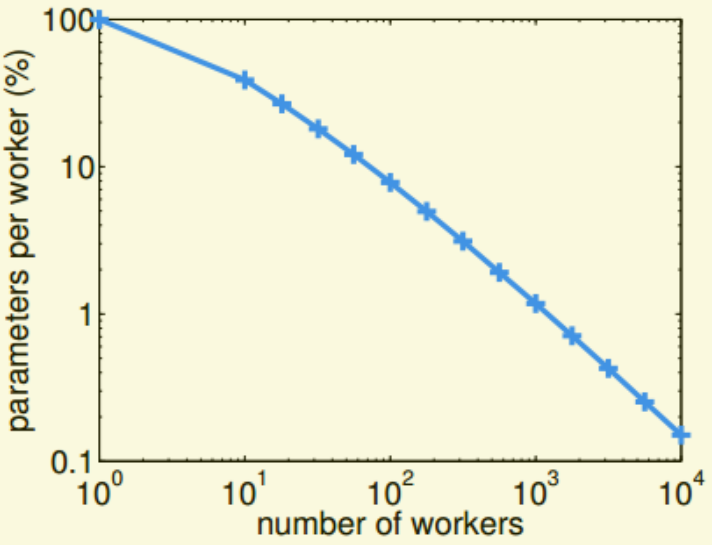


Figure 3: Each worker's set of parameters shrinks as more workers are used, requiring less memory per machine.

worker 越多，每个 worker 上的参数就越少。参数不应该是固定数量的吗？网络是全连接的？

### 3.3 2.3 Generative Models

对于无监督学习，算法 1 需要进行改进，每次跟新不是梯度，而是与文档主题的契合度（topic modeling），同时需要补充信息（单词的含义）。

# 4 3 Architecture

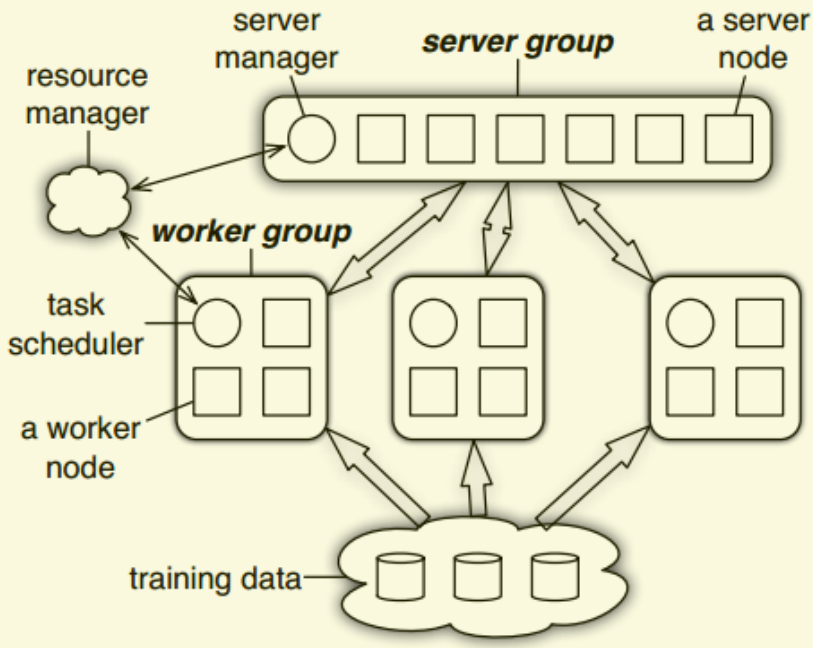


Figure 4: Architecture of a parameter server communicating with several groups of workers.

每个参数服务器既支持单独的参数空间，又支持共享参数空间。

## 4.1 3.1 (Key, Value) vectors

the pair is a feature ID and its weight.

## 4.2 3.2 Range Push and Pull

w.push(R, dest), w.pull(R, dest) 为了编程和效率，支持 range-based push and pull

## 4.3 3.3 User-Defined Functions on the Server

服务器可以执行用户定义的函数

## 4.4 3.4 Asynchronous Tasks and Dependency

- 任务是同步的：
  - The caller marks a task as finished only once it receives the callee's reply.
  - The callee marks a task as finished only if the call of the task is returned and all subtasks issued by this call are finished
- 任务以来能提高算法逻辑性和模型的一致性

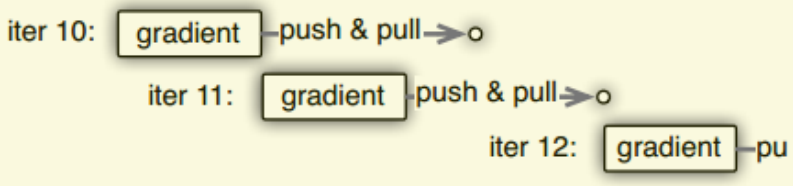


Figure 5: Iteration 12 depends on 11, while 10 and 11 are independent, thus allowing asynchronous processing.

## 4.5 3.5 Flexible Consistency

任务的独立性能提高并行程度，提高系统效率，但是导致不一致性，因此需要 trade-off。

关于任务之间的以来可以建立三种模型：Sequential, Eventual, Bounded Delay。值得注意的是，这些图应该是动态的，因为时间是可变的。

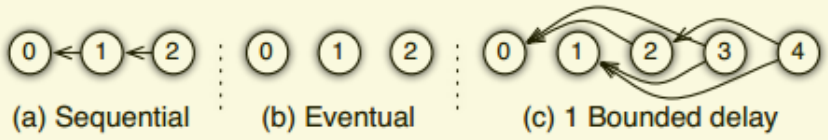


Figure 6: Directed acyclic graphs for different consistency models. The size of the DAG increases with the delay.

### 4.6 3.6 User-defined Filters

使用用户定义的过滤器选择需要同步的 key-value pairs。KKT: a worker only pushes gradients that are likely to affect the weights on the servers

## 5 4 Implementation

### 5.1 4.1 Vector Clock

- 用来记录每个节点 (key, value) pair 的时间。
- 许多参数时间戳相同

ically, assume that $\mathrm{vc}_i(k)$ is the time of key $k$ for node $i$. Given a key range $\mathcal{R}$, the ranged vector clock $\mathrm{vc}_i(\mathcal{R}) = t$ means for any key $k \in \mathcal{R}$, $\mathrm{vc}_i(k) = t$.

**Algorithm 2** Set vector clock to $t$ for range $\mathcal{R}$ and node $i$

1: **for** $\mathcal{S} \in \{\mathcal{S}_i : \mathcal{S}_i \cap \mathcal{R} \neq \emptyset, i = 1, \ldots, n\}$ **do**
2: **if** $\mathcal{S} \subseteq \mathcal{R}$ **then** $\mathrm{vc}_i(\mathcal{S}) \leftarrow t$ **else**
3: $a \leftarrow \max(\mathcal{S}^b, \mathcal{R}^b)$ and $b \leftarrow \min(\mathcal{S}^e, \mathcal{R}^e)$
4: split range $\mathcal{S}$ into $[\mathcal{S}^b, a), [a, b), [b, \mathcal{S}^e)$
5: $\mathrm{vc}_i([a, b)) \leftarrow t$
6: **end if**
7: **end for**

### 5.2 4.2 Messages

$$[\text{vc}(\mathcal{R}), (k_1, v_1), \ldots, (k_p, v_p)] \;\; k_j \in \mathcal{R} \text{ and } j \in \{1, \ldots p\}$$

参数服务器的基本通信格式，我们使用 fast Snappy compression library 压缩信息。

### 5.3 4.3 Consistent Hashing

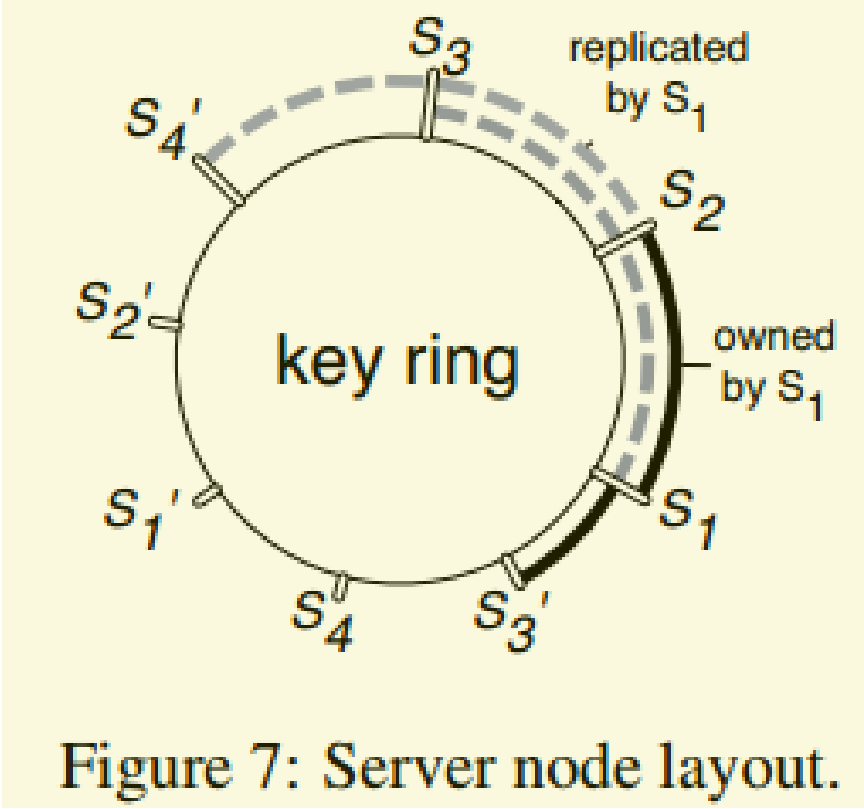


Figure 7: Server node layout.

每个服务器节点负责它的开始到下一个节点，该节点是该 key range 的 master。

### 5.4 4.4 Replication and Consistency

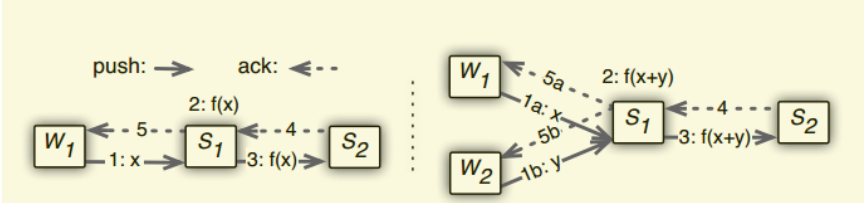


Figure 8: Replica generation. Left: single worker. Right: multiple workers updating values simultaneously.

过多的复制会影响带宽。

### 5.5 4.5 Server Management

为了提高系统容错和动态扩展能力，我们必须支持添加和删除节点。

### 5.6 4.6 Worker Management

添加和删除 worker node

## 6 5 Evaluation

Sparse Logistic Regression and Latent Dirichlet Allocation

### 6.1 5.1 Sparse Logistic Regression

- Problem and Data: ad click prediction dataset, 1000 machines
- Algorithm:

**Algorithm 3** Delayed Block Proximal Gradient [31]

**Scheduler:**

1: Partition features into $b$ ranges $\mathcal{R}_1, \ldots, \mathcal{R}_b$
2: **for** $t = 0$ **to** $T$ **do**
3: Pick random range $\mathcal{R}_{i_t}$ and issue task to workers
4: **end for**

**Worker $r$ at iteration $t$**

1: Wait until all iterations before $t - \tau$ are finished
2: Compute first-order gradient $g_r^{(t)}$ and diagonal second-order gradient $u_r^{(t)}$ on range $\mathcal{R}_{i_t}$
3: Push $g_r^{(t)}$ and $u_r^{(t)}$ to servers with the KKT filter
4: Pull $w_r^{(t+1)}$ from servers

**Servers at iteration $t$**

1: Aggregate gradients to obtain $g^{(t)}$ and $u^{(t)}$
2: Solve the proximal operator

$$w^{(t+1)} \leftarrow \operatorname*{argmin}_u \Omega(u) + \frac{1}{2\eta}\|w^{(t)} - \eta g^{(t)} + u\|_H^2,$$

where $H = \text{diag}(h^{(t)})$ and $\|x\|_H^2 = x^T H x$

它与之前的有四点不同：

- 每次迭代只有一部分参数被更新
- workers 计算梯度和这一部分参数的二阶导数
- 服务器根据聚集的局部梯度求解近似算子来更新模型
- bounded-delay model, Karush-Kuhn-Tucker (KKT) filter

- Results: compare ours with two others, 一些比较结果

### 6.2 5.2 Latent Dirichlet Allocation

用户兴趣，LDA

### 6.3 5.3 Sketches

...

## 7 6 Summary and Discussion

- 提出 PS 架构
- 易用：全局参数共享。高效：所有通信都是同步的。灵活的一致性：在效率和算法的收敛速度折中。
- 弹性扩展和容错