Heterogeneity-aware Distributed Parameter Servers

Jiawei Jiang[1],
Bin Cui[1],
Ce Zhang[2],
Lele Yu[1],

[1]Peking University [2]Department of Computer Science

SIGMOD'17, May 14-19, 2017, Chicago, Illinois, USA
Presented by Ching-Yuan, Tsai

Outline

1. Introduction
2. Preliminaries
3. CONSGD
4. DYNSGD
5. Experiment and Conclusion

Motivation

- The existing systems have bad performance on heterogeneity environments.
    - Network heterogeneity: Data transmission between data centers is normally slower than within data centers.
    - Sharing cluster: Different instances of the same job often have very different execution time.
    - Spot Instances on the Cloud: To minimize cost, a rented cluster often contains different types of instances.

Heterogeneity-aware Distributed Parameter Server

- A novel distributed machine learning system with SGD optimizer.
  - Optimizer: Global learning rate.
    - CONSGD.
    - DYNSGD.
  - Focus on reducing the # updates.

Outline

Destributed SGD

Algorithm 1 SSPSGD [23]

M: # workers, P: # parameter servers, N: # samples, C: # clocks
b: batch size, s: staleness threshold, w_0: initial parameter

Worker $m = 1, ..., M$**:**

Initialize $w_c^m \leftarrow Pull(m, 0)$, $c_p \leftarrow 0$
for $c = 0$ to C:
  $u_c^m \leftarrow 0$
  for $batch = 1$ to $\frac{N}{bM}$:
    $u_c^m \leftarrow u_c^m - \eta \sum_{i=1}^{b} \nabla f(x_i, y_i, w_c^m)$
    $w_c^m \leftarrow w_c^m - \eta \sum_{i=1}^{b} \nabla f(x_i, y_i, w_c^m)$
  $Push(u_c^m)$
  if $c_p < c - s$:
    $(w_{c+1}^m, c_p) \leftarrow Pull(m, c+1)$

Parameter Server $p = 1, ..., P$**:**

Initialize $w \leftarrow w_0$, $c_{min} \leftarrow 0$
function $Push(u_c^m)$:
  $w \leftarrow w + u_c^m$
  if all the workers finish c_{min}:
    $c_{min} \leftarrow c_{min} + 1$
function $Pull(m, c)$:
  if $c < c_{min} + s$:

Instroduction of existing systems

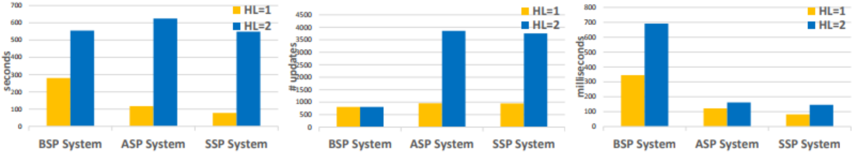
- BSP system: Bulk Synchronous Parallel.
- ASP system: Asynchronous Parallel.
- SSP system: Stale Synchronous Parallel.

HL

- Decomposing the run time of worker into the computation time t_c^m, the transmission time t_t^m.
- The heterogeneous level of the cluster is measured by the speed gap between the fastest weorker and the slowest worker.
    - $HL = \frac{t_c^s + t_t^s}{t_c^f + t_t^f}$

Experiment

- BSP:Spark, ASP:Petuum, SSP:Bosen.
- They activate the sleep() function in 20%workers to simulate the heterogeneous environment.



(a) total run time (b) # updates to converge (c) per-update time

Figure 2: Performance of existing systems in the presence of heterogeneity.

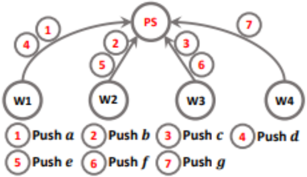
Outline

1. Introduction
2. Preliminaries
3. CONSGD
4. DYNSGD
5. Experiment and Conclusion

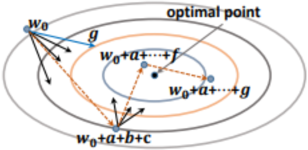
Algorithm

- CONSGD uses a constant global learning rate and multiplies it to each local update.
- $w \leftarrow w + \lambda u_c^m,\ \lambda \in (0, 1)$

Intuition

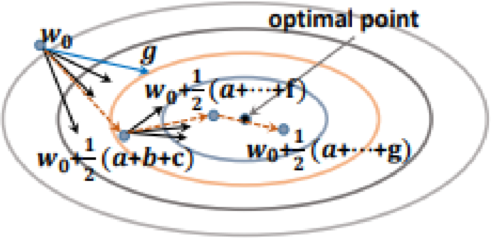


(a) A synchronization example



(b) Why SSPSGD cannot work well

Why CONSGD works better?

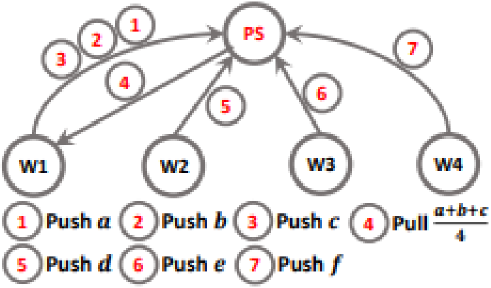


(c) Why CONSGD works better

Choose λ

- $w_{c+1} = \frac{1}{M} \sum_{i=1}^{M} (w_c + u_c^i) = w_c + \frac{1}{M} \sum_{i=1}^{M} u_c^i$
- $\lambda = \frac{1}{M}$ is a good choice.

Limitation



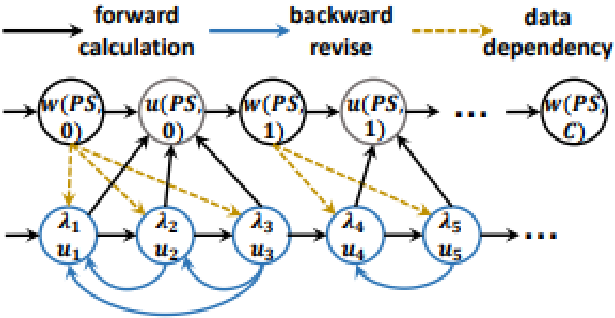
(d) Limitations of ConSGD

Outline

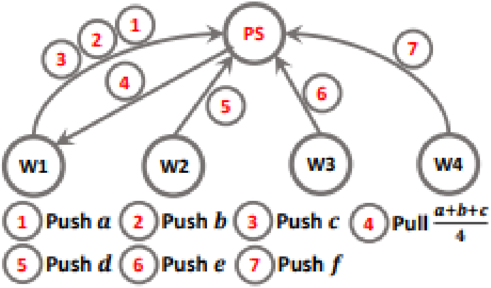
Abstract Model of a Parameter Server

- Denote PS as a tuple $(S, f_{clock}, f_{server}, \lambda, w_0)$,where S is a stream of updates $\{u_1 u_2 ...\}$.
- $staleness(u_i) = |\{u_j : f_{clock}(j) = f_{clock}(i)\}|$.
- $\lambda(i) = \frac{1}{staleness(i)}$
- $w(PS, i) = w_0 + \sum_j \lambda(j) u_j, \forall f_{clock}(j) < i$.
- $u(PS, i) = \sum_j \lambda(j) u_j, \forall f_{clock}(j) = i$.
  - $w(PS, i+1) = w(PS, i) + u(PS, i)$

Intuition



(b) Improved calculation graph of DYNSGD



(d) Limitations of CONSGD

Fast staleness computing

- $V(m)$: the version of the local update from worker m.
- $S(v)$: the staleness of the local update stamped with version v.
- $staleness(u_c^m) = S(V(m))$
- $\Delta u = \frac{1}{d}u_c^m + \frac{d-1}{d}u(PS, v) - u(PS, v) = \frac{1}{d}(u_c^m - u(PC, v))$

Algorithm

Algorithm 2 DynSGD

c_{min}: clock of the slowest worker, c_{max}: clock of the fastest worker

Parameter Server

Initialize $c_{min} \leftarrow 0$, $V \leftarrow 0$, $S \leftarrow 1$, global parameter $w \leftarrow w_0$

function $Push(u_c^m)$:

- $v \leftarrow V(m)$
- $d \leftarrow S(v)$
- $\Delta u = \frac{1}{d}(u_c^m - u(PS, v))$
- $w \leftarrow w + \Delta u$
- $u(PS, v) \leftarrow u(PS, v) + \Delta u$
- $S(v) \leftarrow S(v) + 1$
- $V(m) \leftarrow V(m) + 1$
- **if** all the items in V is larger than v:
  - clear $u(PS, v)$ and $S(v)$
- **if** all the workers finish c_{min}:
  - $c_{min} \leftarrow c_{min} + 1$
- **if** $c + 1 > c_{max}$:
  - $c_{max} \leftarrow c$

function $Pull(m, c)$:

Partition Synchronization

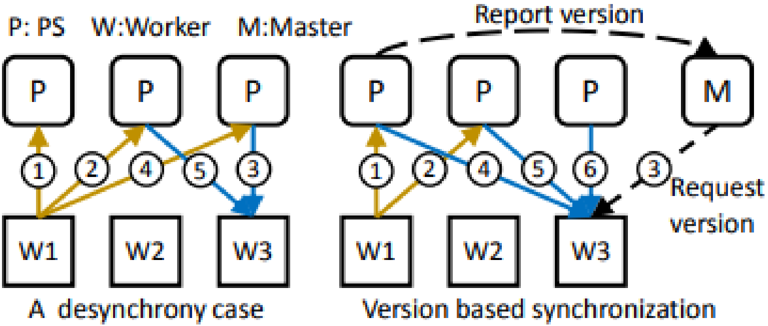


Figure 5: Partition synchronization

Outline

Experiment

Metrics	Setting			Spark (BSP)	Petuum (BSP)	TF (BSP)	Petuum (ASP)	TF (ASP)	Petuum (SSP, s=3)	ConSGD (SSP, s=3)	DynSGD (SSP, s=3)	Petuum (SSP, s=10)	ConSGD (SSP, s=10)	DynSGD (SSP, s=10)
run time	LR	URL	HL=1	280	134	172	117	141	87	85	**82**	78	78	**73**
			HL=2	555	267	347	625	725	250	196	**147**	549	116	**93**
		CTR	HL=1	2352	462	639	748	897	570	456	**342**	646	532	**456**
			HL=2	5198	966	1142	4964	5254	2233	1414	**546**	4501	798	**714**
	SVM	URL	HL=1	188	84	103	88	122	60	55	**51**	62	52	**44**
			HL=2	381	178	226	476	554	188	133	**102**	454	84	**60**
		CTR	HL=1	2829	544	660	480	711	528	539	**360**	544	479	**416**
			HL=2	6040	1104	1436	6120	6944	3690	1907	**873**	6534	1036	**555**
# updates	LR	URL	HL=1	**810**	810	810	955	937	847	862	**833**	949	948	**887**
			HL=2	**810**	810	810	3858	3901	1243	1062	**851**	3756	1144	**891**
		CTR	HL=1	**180**	180	180	317	303	278	229	**164**	492	406	**353**
			HL=2	**180**	180	180	1964	1936	606	404	**204**	2035	517	**453**
	SVM	URL	HL=1	**570**	570	570	826	848	752	687	**628**	969	824	**692**
			HL=2	**570**	570	570	3318	3276	1128	822	**639**	3518	986	**758**
		CTR	HL=1	**240**	240	240	366	393	322	318	**229**	467	447	**387**
			HL=2	**240**	240	240	2757	2781	890	511	**253**	2912	773	**434**
per-update time	LR	URL	HL=1	0.345	0.165	0.212	0.122	0.15	0.103	**0.098**	**0.098**	**0.082**	0.083	**0.082**
			HL=2	0.691	0.33	0.428	0.162	0.185	0.201	0.184	**0.172**	0.146	0.102	**0.1**
		CTR	HL=1	13.0	2.57	3.55	2.36	2.96	2.05	1.99	**1.98**	1.31	1.3	**1.29**
			HL=2	28.88	5.37	6.34	2.53	2.71	3.68	3.5	**3.33**	2.21	1.58	**1.54**
	SVM	URL	HL=1	0.329	0.148	0.18	0.107	0.144	**0.08**	0.081	**0.08**	0.064	0.063	**0.061**
			HL=2	0.659	0.313	0.396	0.143	0.169	0.166	0.162	**0.160**	0.129	0.085	**0.079**
		CTR	HL=1	11.79	2.27	2.75	1.31	1.81	1.64	1.69	**1.57**	1.16	1.07	**1.05**
			HL=2	25.17	4.6	5.98	2.22	2.50	4.15	3.72	**3.45**	2.24	1.34	**1.29**

Conclusion

- They proposed two heterogeneity-aware distributed SGD algorithms under SSP protocol.

Future Work

- Normalizeing updates.