HAL open science

DLBer: A Dynamic Load Balancing Algorithm for the Event-Driven Clusters

Mingming Sun, Changlong Li, Xuehai Zhou, Kun Lu, Hang Zhuang

To cite this version:

Mingming Sun, Changlong Li, Xuehai Zhou, Kun Lu, Hang Zhuang. DLBer: A Dynamic Load Balancing Algorithm for the Event-Driven Clusters. 11th IFIP International Conference on Network and Parallel Computing (NPC), Sep 2014, Ilan, Taiwan. pp.608-611, 10.1007/978-3-662-44917-2_64 . hal-01403163

HAL Id: hal-01403163

https://inria.hal.science/hal-01403163

Submitted on 25 Nov 2016

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution 4.0 International License

DLBer: A Dynamic Load Balancing Algorithm for the Event-driven Clusters

Mingming Sun[1], Changlong Li[1], Xuehai Zhou[2], Kun Lu[1], Hang Zhuang[1]

[1] Computer Science University of Science and Technology of China, Hefei, China
mmsun, liclong, local, zhuangh@mail.ustc.edu.cn
[2] xhzhou@ustc.edu.cn

Abstract. The event-driven programming model has been proposed to efficiently process iterative applications and incremental applications. In clusters based the event-driven model, applications are structured as a series of triggers, each of which will be invoked when associate events are trigged. And framework assigns a newly submitted trigger to a node where the relevant datasets set. Unfortunately it may lead to load imbalance because associate events occur by chance. Numerous triggers in a node may be simultaneously invoked but other nodes have no triggers running. Jobs composed of short, sub-second triggers present a difficult balancing challenge. To the end, we design DLBer, a new dynamic load balancing algorithm for the event-driven clusters to maximize improve the utilization of node resources.

1 Introduction

The synchronous data-flow model such as MapReduce, Dryad and their variants is deficient for iterative applications since synchronous computation and lock-step across rounds. It also is not suitable for incremental applications because it processes total dataset for every increment, incurring a significant performance penalty. Therefore, the event-driven programming model is proposed as an asynchronous computation model such as Percolator[1], Oolongr[2] and Dominor[3] et al. This model follows the Event-Condition-Action (ECA) rule[4]. ECA rules are straightforward: when the event occurs, evaluate the condition; if the condition is fulfilled, execute the action automatically. Applications can be expressed in term of triggers, user-specified code blocks that can be invoked whenever the associated datasets modified. Each trigger completes a computation task. In iterative applications, the result of past iteration can be immediately used to determine the course of current execution. Incremental applications only need to process the updated dataset and some relevant data, which is affected by the updated dataset, instead, recalculates entire dataset.

In clusters based event-driven model, a newly submitted trigger is assigned to the node where the relevant datasets set. However triggers are not instantly executed after their submission, that is to say, specific events are needed to invoke triggers. This may lead to load imbalance because during execution time the cluster resource usage cannot be determined. Extreme case, a node has lots

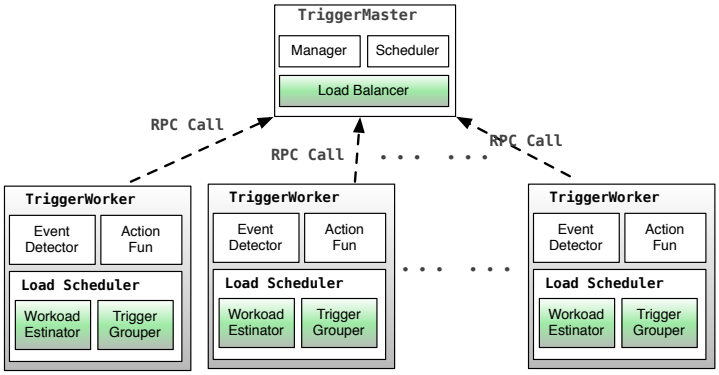

Fig. 1. The architecture of DLBer

of triggers invoked in the waiting queue, but other nodes are idle. So framework needs a load balancer to achieve better utilization of node resources and a high system throughout and quick response time of user requests. However, trigger may be short second or sub-second task. This presents a difficult balancing challenge. To address the above problems, we design DLBer, a new centralized dynamic load balancing algorithm for the event-driven clusters. Dynamic load balancing is essential for such systems since unpredictable load estimates. Our load balancer aims at maximize improve the utilization of node resources.

2 DLBer Design

As shown as Fig. 1, custer based event-driven model runs with a $TriggerMaster$, which assigns a newly coming trigger to workers where the relevant datasets of the trigger set, and multiple $TriggerWorkers$, each of which monitors the modification on dataset, and executes trigger tasks. Our DLBer consists of two main modules: a $LoadBalancer$ in $TriggerMaster$, which coordinates and redistributes triggers invoked among $TriggerWorkers$ according to their workloads, and $LocalScheduler$ in each $TriggerWorker$. $LocalScheduler$ contains $WorkloadEstimator$, which automatically predicts the execution time of trigger and evaluates current total workload, and $TriggerGrouper$, which picks out several suitable triggers to become a group as the basic unit of transfer. As trigger in the event-driven clusters may be short second or sub-second task, we divide triggers invoked into groups to avoid repeatedly network transmission and reduce pressure for $TriggerMaster$. Once the balancing decisions are made in $LoadBalancer$, a high-load $TriggerWorker$ will receive transfer instructions and trigger $LocalScheduler$ to transfer triggers. $LocalScheduler$ first selects one or more trigger group, and then transfer trigger group to low-load destination $TriggerWorker$ according to transfer instructions. DLBer employs centralized

load balancing policy because the accumulation of workoad information can be achieved by heartbeat message.

2.1 Trigger Group Policy

$WorkloadEstimator$ calculates the mean completed time of same triggers, which have already completed, as the execution time of trigger. As triggers continue to complete, the execution time of trigger will be recalculated to get more precise time. Each $TriggerWorker$ workload is the sum of the execution time of all triggers invoked in waiting queue.

The role of $TriggerGrouper$ is to divide triggers invoked into groups as the basic unit of transfer. We define $TriGroup$, the size of trigger group, which depends on the worker computing capacity and network bandwith. Upon the arrival of a transfer request from $TriggerMaster$, $TriggerGrouper$ sorts triggers in descending order of predicted execution time into trigger sequence. Then it selectes triggers from head of trigger sequence into a trigger group according to $TriGroup$. Then trigger group contains minimal triggers to reduce network transmission overhead.

2.2 Load Balancing Algorithm

DLBer focuses on $TriggerWorkers$, whose workload is lower than a threshold called $tunder$. $tunder$ is greater than the sum of load balancing policy time, the heartbeat interval (default as 5 second) and transmission time of a trigger group. Then the low-load $TriggerWorker$ can be still busy during the transfer triggers.

Algorithm 1 illustrates the process of load balancing algorithm. If workload of worker n is below the $tunder$, load balancing policy first considers the heaviest load worker a. Supposing worker a will transfer out a trigger group, the workload of worker a needs to exceed the load of worker n, otherwise it is not necessary to transfer tasks. When all nodes in cluster are busy, there is almost no extra scheduling overhead. At the other extreme, our method also avoid frequent task transfering since the defined $tunder$.

3 Evaluation

In order to test DLBer using a realistic workload, we ported Domino by writing a Domino load balancing plugin. Domino is an open-source trigger-based programming framework. We use application PageRank on a cluster, which has a master and 8 workers. PageRank in the event-driven model contains two triggers: $PageRankDist$ and $PageRankSum$. Once pagerank value of a page is modified, $PageRankDist$ will be invoked to change pagerank weights for all its relevant out-degree pages. Then the $PageRankSum$ for such relevant out-degree pages will be invoked for these changes. The function of $PageRankSum$ is to simply sum up all the rank values generated from different in-degree pages.

Algorithm 1 : *Load Balancer*

Input:
Array A: nodes queue by descending workload;
Iteration:
while $n.load < tunder$ (n : *the tail of* A) **do**
if $a.load > tunder$ & $a.load - TriGroup > n.load + TriGroup$ (a : *the head of* A) **then**
$task_fetch(a, n)$;
else
break;
end if
Reorder array by descending workers' workload;
end while

PageRankDist will be invoked again. This continues until a tolerable error defined by user. Our results shows our load balancer outperforms default Domino about 100%.

Acknowledgment

Our work was supported by the National Science Foundation of China under grants No. 61272131 and No. 61202053, China Postdoctoral Science Foundation grant No. BH0110000014, Fundamental Research Funds for the Central Universities No. WK0110000034, and Jiangsu Provincial Natural Science Foundation grant No. SBK201240198.

References

1. Peng, Daniel, and Frank Dabek. "Large-scale Incremental Processing Using Distributed Transactions and Notifications." OSDI. Vol. 10. (2010)
2. Mitchell, Christopher, Russell Power, and Jinyang Li. "Oolong: asynchronous distributed applications made easy." Proceedings of the Asia-Pacific Workshop on Systems. ACM (2012)
3. Dai, Dong, et al. "Domino: an incremental computing framework in cloud with eventual synchronization." Proceedings of the 23rd international symposium on High-performance parallel and distributed computing. ACM (2014)
4. McCarthy, Dennis, and Umeshwar Dayal. "The architecture of an active database management system." In ACM Sigmod Record, vol. 18, no. 2, pp. 215-224. ACM, (1989)
5. Willebeek-LeMair, Marc H., and Anthony P. Reeves. "Strategies for dynamic load balancing on highly parallel computers." Parallel and Distributed Systems, IEEE Transactions on 4, no. 9: 979-993 (1993)