DOI：10.7652/xjtuxb201710010

通信密集环境下基于内存利用率的预计算方法

刘强[1]，董小社[1]，陈衡[1]，王寅峰[2]
(1.西安交通大学电子与信息工程学院，710049，西安；2.深圳信息职业技术学院软件学院，518172，广东深圳)

摘要：针对通信密集型图计算环境下原静态最大消息数阈值方法由于内存不足导致的频繁低效I/O问题，提出了一种基于内存利用率的预计算方法。该方法利用了图应用的计算满足交换律和结合律的特点，根据当前进程的内存利用率判断是否将本轮超步通信过程中的部分消息进行预计算，同时在预计算过程中使用细粒度锁以增大预计算线程的并发度；在下轮超步的正常计算时合并上轮的预计算结果，实现了通信和计算的重叠，达到减少作业响应时间和磁盘I/O开销的目的。实验结果表明，在通信密集场景下，该方法在性能和I/O开销上均优于已有的MMT方法，作业响应时间减少了5.9%～79.0%，同时计算过程中的磁盘开销减少了9.99%～79.87%。

关键词：通信密集型图计算；内存利用率；预计算
中图分类号：TP391 **文献标志码**：A **文章编号**：0253-987X(2017)10-0059-06

A Precomputation Method Based on Memory Utilization in Intensive Communication Environments

LIU Qiang[1], DONG Xiaoshe[1], CHEN Heng[1], WANG Yinfeng[2]
(1. School of Electronic & Information Engineering, Xi'an Jiaotong University, Xi'an 710049, China;
2. School of Software, Shenzhen Institute of Information Technology, Shenzhen, Guangdong 518172, China)

Abstract: A precomputation method based on memory utilization (MUP) is proposed to improve the I/O-inefficient problem caused by limited memory of the static maximum message threshold method (MMT) in intensive communication environments. The method leverages both the associative law and commutative law of graph computations, and precomputes some partial incoming messages of current super-step based on the process memory utilization. The precomputed results are then combined in the next super-step's normal computations. Moreover, a find-grained lock mechanism is used to increase the parallel granularity of precomputation threads. The method reduces the job response time and the expensive disk I/O costs incurred by messages through overlapping computation and communication. Experimental results show that MUP is better than the original MMT method in both the performance and I/O costs. The job response time is improved by 5.9%～79.0% and high redundant disk I/O costs are reduced by 9.99%～79.87% in intensive communication environments.

Keywords: graph computing; memory utilization; precomputation

---

收稿日期：2017-03-07。 作者简介：刘强(1985—)，男，博士生；董小社(通信作者)，男，教授，博士生导师。 基金项目：国家重点研发计划资助项目(2016YFB0201402，2016YFB0201800)；国家自然科学基金资助项目(61572394)；深圳市科技计划资助项目(JSGG20140519141854753)。
网络出版时间：2017-08-19 网络出版地址：http://www.cnki.net/kcms/detail/61.1069.T.20170819.1125.010.html

近年来,随着社交网络分析、数据挖掘、生物基因图谱等新兴应用的大量涌现,其对应的图数据规模呈显著增长趋势。这类应用的数据具有顶点数量多,顶点间链接关系复杂等特点。截至 2015 年 3 月,Twitter 拥有超过 2.88 亿活跃用户以及超过 600 亿条边组成的用户关系图;Facebook 有 13.9 亿活跃用户及超过 4 000 亿条边[1]。随着图数据的快速增长,一系列具有代表性的分布式大图处理框架也相继涌现,例如 Pregel[2]、X-Stream[3]、Power Grpah[4]等。

大规模图数据处理具有多轮迭代、实时性强等特点,要求数据在计算过程中尽可能存储在内存中以提高性能。现有的分布式集群中,与 CPU 的计算能力相比,内存容量等性能指标提升仍显不足,增长速度赶不上数据规模的增长速度,导致内存不足的情况出现,甚至在一些 Top500 超级计算机系统中也存在同样情况[5]。近年来固态硬盘(solid state disk,SSD)、混合内存[6]等新技术仍存在价格昂贵、使用能耗较高、寿命较短[7]等缺点,不能很好地满足图应用的需求。特别是对于通信密集型图应用,其多轮迭代产生的大量中间消息以及相应的检查点数据存储开销,给集群内存造成了巨大压力,同时产生大量磁盘 I/O 开销,导致严重的性能损耗[8]。因此,在内存受限的情况下研究如何减少图作业特别是通信密集型作业的磁盘 I/O 开销,成为图计算中的热点问题。

为了解决图计算中由于内存不足导致的频繁磁盘 I/O 问题,Kyrola 等提出了一种基于磁盘的单机大图处理系统 GraphChi[9]。GraphChi 采用并行滑动窗口(parallel sliding windows,PSW)技术以减少对磁盘的随机访问,但其在预处理阶段需要对边的源顶点进行排序增加了额外开销,同时没有充分利用数据加载和计算的并行以提高性能,作为一种单机图计算框架,GraphChi 也存在可扩展性受限等问题。Pregelix 引入了数据库理论中“表连接”的概念,将顶点和消息的计算过程抽象成表元组的连接操作,并通过对大规模顶点数据建立 B 树索引的方法以减轻磁盘开销[10],但其建立和维护 B 树索引同样需要额外的时间和空间开销。一些研究通过对图的存储结构进行优化以减少内存开销,如 GPS 通过消息合并、对象序列化和对图顶点及消息进行数据归约的方式[11]以节省内存资源;Ligra+通过对图数据进行压缩[12]以缓解内存压力,但是这些方法均需要在通信阶段保存大量消息在内存中,导致资源利用不足。

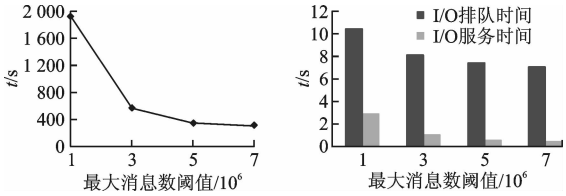
Pregel 是 Google 公司于 2010 年提出的一种基于以顶点为中心的大同步(bulk synchronous parallel,BSP)[13]模型设计的并行图计算框架,将计算分解为一系列的超步,顶点间通过消息进行通信。在现有的 Pregel 的开源实现(如 Giraph)中,通常采用静态最大消息数阈值(max message threshold,MMT)的方法来限制中间消息的内存占用量,当中间消息数超过阈值时,将多余消息写入磁盘,并在后续计算中读取,但是该方法没有考虑图计算应用的特点以及分布式环境下资源可用性动态变化的情况。对于计算密集型应用,由于其计算过程中产生的消息量较少,因而不会出现中间数据过大导致频繁磁盘 I/O;但是对于通信密集型应用,由于其通信量大、多次迭代的特点,频繁的磁盘 I/O 会严重影响作业性能。静态的最大消息数阈值并不能很好地适应图应用和系统中资源可用性动态变化的特点,进而影响系统性能。

针对通信密集型应用场景下原 MMT 方法由于内存不足导致的中间数据频繁低效 I/O 的问题,本文提出了基于内存利用率的预计算方法(memory utilization based precomputation method,MUP)。该方法利用图应用计算满足交换律和结合律的特点,通过对内存利用率进行判断,将原方法中写入磁盘的部分消息提前计算,并在正常计算时将两者结果合并的方法,实现了通信和计算的重叠,进而减轻磁盘 I/O 开销,提升系统性能。

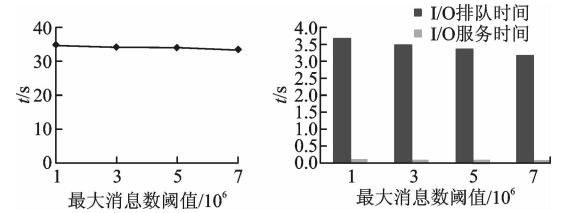
1 问题分析

为了测试通信密集型作业和普通作业在 MMT 方法下,不同静态消息数阈值对磁盘 I/O 开销和性能的影响,本文选取 Giraph 自带测试用例中的单源最短路径算法(SSSP)和三角形计数算法(Triangle)进行测试。使用 linux 下的 iostat 命令来测试磁盘开销,主要选择具有代表性的 await 和 svctm 这 2 个指标,其中 await 代表任务进行 I/O 操作的平均排队时间;svctm 代表任务进行 I/O 操作的平均服务时间。输入数据集为 live-journal,并行任务数分别为 6 和 8,任务可用内存均设置为 1.5 GB,最大消息数阈值在 $10^6 \sim 7\times10^6$ 范围内变化,集群规模为 6 个节点。测试结果如图 1 所示,由图 1a、图 1b 可知,通信密集型作业(例如 Triangle)由于中间消息量大,导致 I/O 开销较大,严重影响了系统性能。随着消息数阈值的增加,作业响应时间和 I/O 开销

均呈递减趋势，逐渐逼近性能瓶颈。对于普通作业（例如 SSSP）而言，消息数阈值变化对性能和 I/O 开销影响则较小，如图 1c 和图 1d 所示。



(a)通信密集型作业响应时间　(b)通信密集型作业磁盘 I/O 开销



(c)普通作业响应时间　(d)普通作业磁盘 I/O 开销

图 1　作业响应时间和磁盘 I/O 开销随最大消息数阈值的变化

由图 1 的测试结果可知，消息数阈值的增加对通信密集型作业的性能和 I/O 开销均存在局限性，且静态的阈值设置方式并不能很好地适应作业动态变化的需求，造成计算资源的浪费，因此提出一种基于内存利用率的预计算方法，根据图应用和内存可用性动态变化特点对部分消息提前计算，以减少磁盘 I/O 开销并提高系统吞吐率。

2　基于内存利用率的预计算方法

2.1　图的预计算模型

为了描述需要，本文对系统进行如下定义：设输入图 $G=(V,E)$，其中 V 代表顶点集合，E 代表边集合。对于 $\forall v\in V$，设 $B(v_i)$ 代表点 v 在第 i 轮超步时的值，令 F 代表 Pregel 模型下每轮超步中图应用的计算函数，$M(v_i)$ 代表点 v 在第 i 轮超步时收到总的消息集合，则图应用每轮超步时的正常计算过程可表示为

$$B(v_{i+1})=F(B(v_i),M(v_i)) \tag{1}$$

在并行计算场景下，图 G 被切分为 k 个分区，分别设为 $P_1,P_2,\cdots,P_k(1\leqslant k\leqslant|E|)$，则总消息集合 $M(v_i)$ 也相应的被划分为 $M_{v_i}^{P_1},M_{v_i}^{P_2},\cdots,M_{v_i}^{P_k}$。其中，$M_{v_i}^{P_k}$ 代表第 i 轮超步时点 v 收到的来自分区 P_k 的消息集合。如果 F 满足交换律和结合律，则点 v 的计算过程可表示为

$$F(B(v_i),M(v_i))=C(F(B(v_i),M_{v_i}^{P_1}),F(B(v_i),M_{v_i}^{P_2}),\cdots,F(B(v_i),M_{v_i}^{P_k})) \tag{2}$$

式中：C 代表值合并运算，设预计算开始时点 v 已经收到的分区消息集合为 N_1，尚未到达的分区消息集合为 N_2，即 $N_2=M(v_i)-N_1$。如果 F 满足交换律和结合律，则预计算模型可以表示为

$$B(v_{i+1})=C(F(B(v_i),C(\bigcup_{P_k\in N_1}M_{v_i}^{P_k})),F(B(v_i),C(\bigcup_{P_k\in N_2}M_{v_i}^{P_k}))) \tag{3}$$

以图遍历类算法中经典的 SSSP 算法为例，设点 v 在第 i 轮超步时收到的各分区消息分别为 $M_{v_i}^{P_1},M_{v_i}^{P_2},\cdots,M_{v_i}^{P_k}$，则该算法每轮超步时的正常计算过程可表示为

$$B(v_{i+1})=B(v_i)+\min(M_{v_i}^{P_1},M_{v_i}^{P_2},\cdots,M_{v_i}^{P_k}) \tag{4}$$

其中，min 函数代表取较小值运算，由于 min 函数满足交换律和结合律，则由式(1)可知，原计算过程可以表示为

$$B(v_{i+1})=\min(B(v_i)+M_{v_i}^{P_1},B(v_i)+M_{v_i}^{P_2},\cdots,B(v_i)+M_{v_i}^{P_k}) \tag{5}$$

设 $B'(v_{i+1})$ 代表采用预计算方法后，点 v 在第 $i+1$ 轮超步时的值，则结合式(3)可将预计算过程表示为

$$B'(v_{i+1})=\min(B(v_i)+\min(\bigcup_{P_k\in N_1}M_{v_i}^{P_k}),B(v_i)+\min(\bigcup_{P_k\in N_2}M_{v_i}^{P_k})) \tag{6}$$

根据 min 函数性质可知，$B(v_{i+1})=B'(v_{i+1})$，即如果图计算满足交换律和结合律，则消息的到达顺序并不会影响 SSSP 算法的计算结果。除了图遍历类算法，其他图算法如以 Pagerank 为代表的页面权重类算法，由于满足交换律和结合律[14]，因此同样可以使用本文方法进行优化计算以提高性能。

2.2　基于内存利用率的预计算方法

如果一个图应用的计算满足交换律和结合律，就可以采取预计算的方法进行加速，MUP 的总体设计架构如图 2 所示。

当新的消息到来时，如果 Java 虚拟机（Java virtual machine，JVM）的内存利用率超过阈值，则对收到的所有消息按分区消息数由大到小进行排序，然后启动预计算线程对这部分消息进行预计算，并将结果存储在缓存池中。当下一轮超步的正常计算开始时，对于已经完成部分预计算的顶点，需要根据应用的值合并函数合并预计算结果，其余顶点则执行正常计算。为了避免预计算线程并发计算时产生数据冲突，通过对顶点计算函数加锁的方法以保证

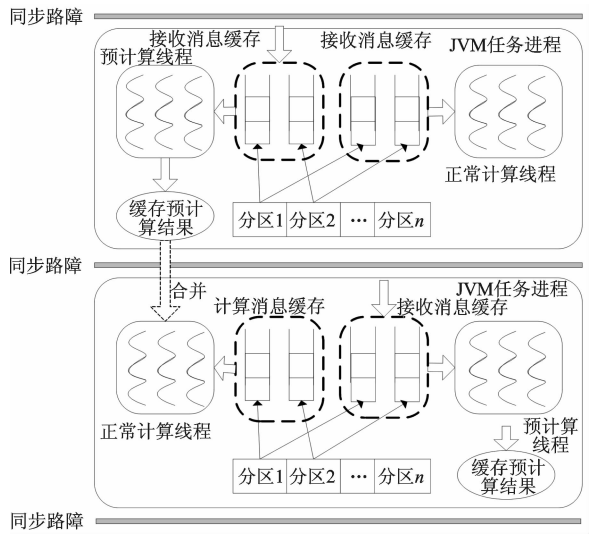


图 2 MUP 总体设计架构

计算结果的正确性和一致性。

参考 Java 虚拟机的堆存储内存设计[15]，设 JVM 当前内存利用率 J_u 为

$$J_u = (J_t - J_f)/J_m \tag{7}$$

式中：J_t 代表 JVM 当前所占用总的堆内存；J_f 代表 JVM 当前的空闲堆内存；J_m 代表当前操作系统给 JVM 分配的最大堆内存。

为了简化分析，这里假定任务各分区的消息数是均匀的，当内存超过阈值时，设顶点总数为 $|V|$，当前分区数为 k，则采用快速排序算法对分区消息数排序的开销为 $O(k\mathrm{lb}k)$，设分区内平均每个顶点的消息数为 m，每个顶点的计算函数时间开销为 T_F，则分区的预计算线程时间开销为 $O(m(|V|/k)T_F)$。由于这些消息分区是并发执行的，因此线程并发竞争引起的锁的时间开销与预计算分区数相关，因此设锁的时间开销为 $L(k)$，则执行单个预计算线程的时间复杂度为 $O(k\mathrm{lb}k)+O(m(|V|/k)T_F)+L(k)$。

设正常计算时两次超步间的总时间开销为 T，同步和通信的总时间开销为 T_c，需要写磁盘的消息分区数为 P_1，留在内存中的消息分区数为 P_2，P_1 的读写磁盘开销为 T_{IO}，则有

$$T = T_c + T_{IO}(P_1) + T_F(P_1 + P_2) \tag{8}$$

设采用预计算方法后两次超步间的总时间开销为 T'，预计算开销为 T_p，合并预计算结果的开销为 T_b，则预计算总开销为

$$T' = \max(T_c, T_p(P_1)) + T_F(P_2) + T_b(P_1) \tag{9}$$

由式(9)可知，预计算过程与通信过程是重叠的，不会带来较大的额外时间开销。因此，根据式(8)、式(9)可知，$\max(T_c, T_p(p_1)) \leqslant T_{IO}(P_1) + T_c$，对于 P_1 的消息分区，由于已经完成计算，其结果合并的时间开销小于原方法中的重新计算消息的开销，从而有 $T_F(P_2) + T_b(P_1) \leqslant T_F(P_1 + P_2)$，因此 $T' \leqslant T$。与原方法相比，基于内存利用率的预计算方法实现了计算与通信阶段的重叠，不但减少了 CPU 的等待时间，同时也避免了原 MMT 方法产生的 I/O 开销。

3 实验验证

实验集群由 6 台刀片节点构成，节点配置为 2×Intel Xeon E5-2670(2.6 GHz，8 核处理器)，32 GB 内存，300 GB 硬盘，采用千兆以太网互连，实验软件为 Giraph1.1.0 版本。

3.1 测试用例及数据

本实验使用的测试用例采用来自 Giraph 自带的测试用例集，包括 Triangle、Pagerank、MinDistance 和 SSSP，每组作业测试 3 次。在测试过程中，重点关注各测试用例中作业的平均响应时间和磁盘 I/O 开销。本文通过统计作业中任务的 I/O 操作的平均排队时间和服务时间来衡量磁盘 I/O 开销。测试数据集 1 和数据集 2 均来自 SNAP(斯坦福公共数据集，http://snap.stanford.edu/)，其参数见表 1。

表 1 图数据集的顶点数和边数

图数据集	顶点数	边数
数据集 1	4 847 571	68 993 773
数据集 2	3 072 441	117 185 083

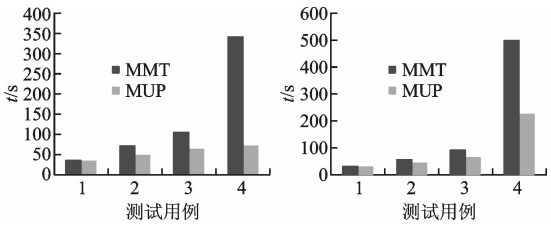
以上 4 个测试用例中，Triangle 是统计顶点间三角数关系用例，Pagerank 是网页分级排名用例，MinDistance 是求最小顶点 ID 用例，SSSP 是求单源最短路径用例。将应用的通信密集程度按从小到大排列依次为：SSSP，Mindistance，Pagerank，Triangle。测试过程中，为了模拟内存不足的情况，除了 Triangle 用例外，其余用例的内存用量均设置为 1 GB，内存最大消息数阈值均使用默认的 10^6。为了避免由于内存不足而造成的 JVM 进程崩溃，将 Triangle 的测试内存增至 1.5 GB，最大消息数阈值提至 5×10^6 以使消息尽量多的保存在内存中，同时内存利用率阈值设置为 0.875。测试用例的并行任务数、任务内存开销及网络通信量见表 2。

表 2　测试用例的并行任务数、子任务内存及网络通信量

测试用例	并行任务数		内存/GB		网络通信量/B	
	数据集 1	数据集 2	数据集 1	数据集 2	数据集 1	数据集 2
1	6	10	1	1	1 091 000 483	1 875 421 038
2	6	12	1	1	2 039 643 602	5 191 670 246
3	8	12	1	1	5 520 515 600	9 377 306 480
4	8	45	1.5	2	7 462 961 112	88 307 352 547

3.2　测试结果

性能测试结果如图 3 所示,结合表 2、图 3 可知,随着图应用通信密集程度的加深,其增大的消息数会造成更加频繁的磁盘 I/O 开销,从而导致 MUP 的性能改进呈增长趋势,作业响应时间减少了 5.9%～79.0%。其中,SSSP 改进最小,Triangle 用例的通信最频繁,改进也最大。此外,在数据集 1 下的性能改进要高于数据集 2,如 Triangle 用例在数据集 1 下的作业响应时间减少了 79.0%,在数据集 2 下只减少了 54.7%,这是因为根据 Giraph 默认的哈希顶点划分算法,其分区的总量为并行任务数的平方,根据表 2 中通信量和并行任务数可知,对于每个分区而言,数据集 1 下平均每个分区的消息数大于数据集 2,因此原方法的 I/O 开销更大,导致 MUP 加速效果更好。

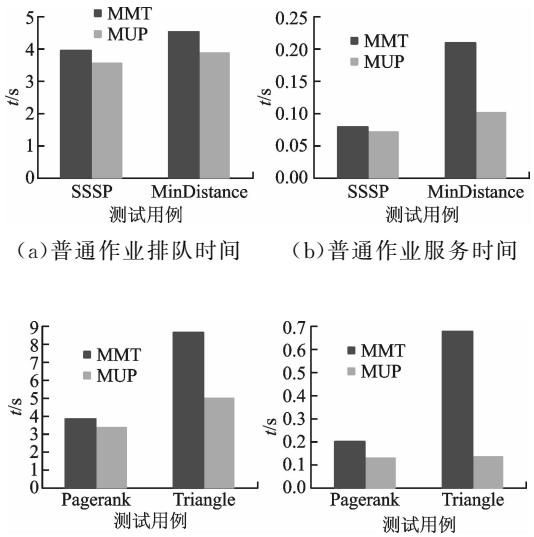


(a)数据集 1　　(b)数据集 2

图 3　作业响应时间

数据集 1 下的各作业任务的平均磁盘开销对比如图 4 所示,对于通信频度较低的应用,例如 SSSP,与原方法相比,MUP 在 I/O 排队时间和服务时间上分别减少了 9.99%和 10.53%,而对于通信量较大的 MinDistance 应用,排队时间和服务时间分别减少了 14.07%和 51.48%。图 4c 和图 4d 分别给出了在通信密集型图应用(Pagerank 和 Triangle)下 I/O 开销的对比,可知 2 种应用的磁盘开销均显著减少,尤其是 Triangle 测试用例,其任务排队时间和服务时间分别减少了 42.32%和 79.87%。

综上所述,在通信密集场景下,与原方法相比,MUP 能够显著减少原方法中频繁的 I/O 开销,提升系统性能。



(a)普通作业排队时间　(b)普通作业服务时间

(c)通信密集作业排队时间　(d)通信密集作业服务时间

图 4　磁盘 I/O 开销

4　结　论

本文提出了基于内存利用率的预计算方法,该方法利用了图应用计算满足交换律和结合律的特点,对需要写入磁盘的消息进行了预计算以减轻磁盘 I/O 开销,同时通过预计算和通信的重叠以提高系统性能。实验结果表明,MUP 方法能够有效减少作业响应时间,避免 CPU 等计算资源的浪费,在作业响应时间和资源利用率等方面优于已有的静态最大消息数阈值方法。下一步的工作是将该方法应用于多种并行图计算框架,例如 GraphX[16] 和 PowerGraph 等,从通信和调度等方面,利用消息剪枝等策略进一步完善该优化方法。

参考文献:

[1] CHING A, EDUNOV S, KABILJO M, et al. One trillion edges: graph processing at facebook-scale [J]. Proceedings of the VLDB Endowment, 2015, 8(12): 1804-1815.

[2] MALEWICZ G, AUSTERN M H, BIK A J C, et al. Pregel: a system for large-scale graph processing [C]

// Proceedings of the 41st ACM Sigmod International Conference on Management of Data. New York, USA: ACM, 2010: 135-146.

[3] ROY A, MIHAILOVIC I, ZWAENEPOEL W. X-stream: edge-centric graph processing using streaming partitions [C] // Proceedings of the 24th ACM Symposium on Operating Systems Principles. New York, USA: ACM, 2013: 472-488.

[4] GONZALEZ J E, LOW Yucheng, GU Haijie, et al. PowerGraph: distributed graph-parallel computation on natural graphs [C]// Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation. Berkeley, CA, USA: USENIX Association, 2012: 17-30.

[5] SHI Xuanhua, CHEN Ming, HE Ligang, et al. Mammoth: gearing hadoop towards memory-intensive mapreduce applications [J]. IEEE Transactions on Parallel and Distributed Systems, 2015, 26(8): 2300-2315.

[6] DHIMAN G, AYOUB R, ROSING T. PDRAM: a hybrid PRAM and DRAM main memory system [C]// Proceedings of the 46th Annual Design Automation Conference. New York, USA: ACM, 2009: 664-669.

[7] 罗乐, 刘轶, 钱德沛. 内存计算技术研究综述 [J]. 软件学报, 2016, 27(8): 2147-2167.
LUO Le, LIU Yi, QIAN Depei. Survey on in-memory computing technology [J]. Journal of Software, 2016, 27(8): 2147-2167.

[8] WANG Zhigang, GU Yu, BAO Yubin, et al. Hybrid pulling/pushing for I/O-efficient distributed and iterative graph computing [C]// Proceedings of the 47th International Conference on Management of Data. New York, USA: ACM, 2016: 479-494.

[9] KYROLA A, BLELLOCH G, GUESTRIN C. GraphChi: large-scale graph computation on just a PC [C]// Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation. Berkeley, CA, USA: USENIX Association, 2012: 31-46.

[10] BU Yingyi, BORKAR V, JIA Jianfeng, et al. Pregelix: big(ger) graph analytics on a dataflow engine [J]. Proceedings of the VLDB Endowment, 2014, 8(2): 161-172.

[11] MCCUNE R R, WENINGER T, MADEY G. Thinking like a vertex: a survey of vertex-centric frameworks for large-scale distributed graph processing [J]. ACM Computing Surveys, 2015, 48(2): 1-39.

[12] SHUN Julian, DHULIPALA L, BLELLOCH G E. Smaller and faster: parallel processing of compressed graphs with Ligra+ [C]// Proceedings of the 25th Data Compression Conference. Piscataway, NJ, USA: IEEE Computer Society, 2015: 403-412.

[13] VALIANT L G. A bridging model for parallel computation [J]. Communications of the ACM, 1990, 33(8): 103-111.

[14] ZHANG Yanfeng, GAO Qixin, GAO Lixin, et al. Maiter: an asynchronous graph processing framework for delta-based accumulative iterative computation [J]. IEEE Transactions on Parallel and Distributed Systems, 2014, 25(8): 2091-2100.

[15] SHIRAZI J. Java performance tuning [M]. Sebastopol, CA, USA: O'Reilly & Associates Inc., 2000: 45-49.

[16] GONZALEZ J E, XIN R S, DAVE A, et al. GraphX: graph processing in a distributed dataflow framework [C]// Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation. Berkeley, CA, USA: USENIX Association, 2014: 599-613.

[本刊相关文献链接]

张保,曹海军,董小社.面向图形处理器重叠通信与计算的数据划分方法.2011,45(4):1-4.[doi:10.7652/xjtuxb201104001]

李健,黄庆佳,刘一阳,等.云计算环境下的大规模图状数据处理任务调度算法.2012,46(12):116-122.[doi:10.7652/xjtuxb201212020]

庄威,桂小林,林建材.云环境下基于多属性层次分析的虚拟机部署与调度策略.2013,47(2):28-32.[doi:10.7652/xjtuxb201302005]

鲍军鹏,宋楠,陶斌.依赖知识的服务组合算法.2013,47(8):1-6.[doi:10.7652/xjtuxb201308001]

李亮,王恩东,朱正东.应用动态生成树的 GPU 显存数据复用优化.2013,47(10):44-50.[doi:10.7652/xjtuxb201310008]

熊金波,姚志强,马建峰,等.云计算环境中的组合文档模型及其访问控制方案.2014,48(2):25-31.[doi:10.7652/xjtuxb201402005]

马莉,唐善成,王静,等.云计算环境下的动态反馈作业调度算法.2014,48(7):77-82.[doi:10.7652/xjtuxb201407014]

郑鹏飞,尤佳莉,王劲林,等.一种多租户云的内部网络共享策略.2014,48(8):54-59.[doi:10.7652/xjtuxb201408010]

孙霞,张敏超,冯筠,等.Hadoop 框架下的多标签传播算法.2015,49(5):134-139.[doi:10.7652/xjtuxb201505021]

周墨颂,朱正东,董小社,等.采用资源划分的云环境下 Hadoop 资源许可调度方法.2015,49(8):69-74.[doi:10.7652/xjtuxb201508012]

(编辑 武红江)