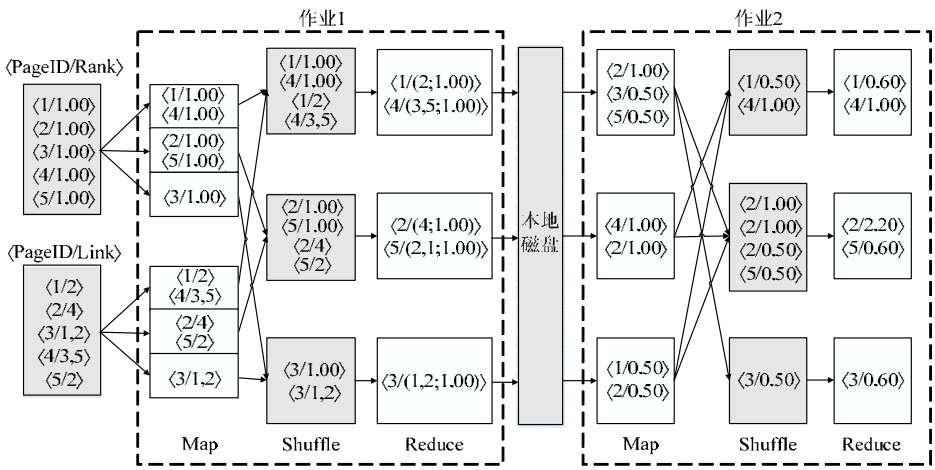

Fig.9 MapReduce for PageRank with one iteration

图 9 MapReduce 处理 PageRank 问题一次迭代

由图 9 可以看到,在 MapReduce 一次迭代过程中,需要注意以下几点.

- 与 WordCount 类似,MapReduce 中每个 Map 任务都需要在输出数据前对中间数据进行排序,使其分区内有序;
- 每一个 Map 节点与 Reduce 节点单独地作为一个任务,不同任务之间独立计算互不干扰,每一个任务的计算结果都要存入磁盘进行数据传输(图 9 中所有阴影部分表示需要存入磁盘的数据);
- 为了保证系统的可靠性和可扩展性,MapReduce 要求所有的 Reduce 需要在所有 Map 任务结束后才可以进行计算;并且只有在作业 1 所有任务完成后,作业 2 才能开始运行任务.

MapReduce 中,PageRank 的两次迭代过程被划分为 4 个作业进行计算,共有 23 个任务、4 次 shuffle 操作. 图 10 为 MapReduce 处理 PageRank 问题两次迭代的流程图.

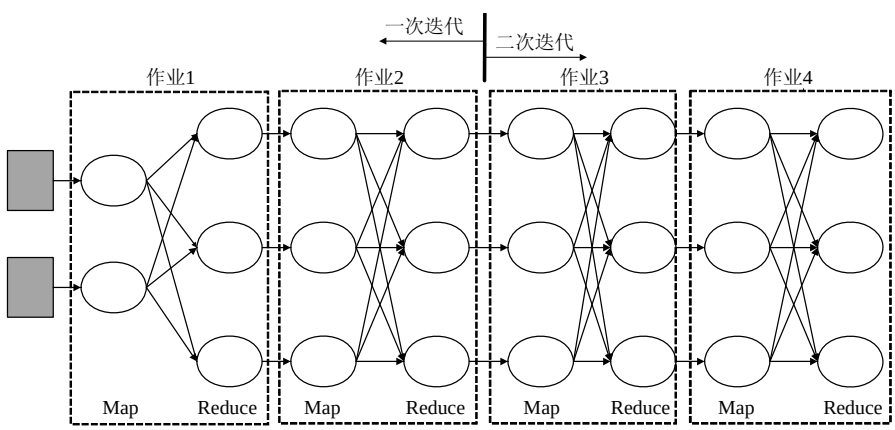

Fig.10 MapReduce for PageRank with two iterations

图 10 MapReduce 处理 PageRank 问题两次迭代

5.3 Spark处理PageRank问题

Spark 在处理迭代问题时,相较于 MapReduce 做了很多改进:首先,在内存足够的情况下,Spark 允许用户将常用的数据缓存到内存中,加快了系统的运行速度;其次,Spark 对数据之间的依赖关系有了明确的划分,根据宽依赖与窄依赖关系进行任务的调度,可以实现管道化操作,使系统的灵活性得以提高.

图 11 与图 12 分别是 Spark 处理 PageRank 问题的一次迭代与两次迭代的流程图.

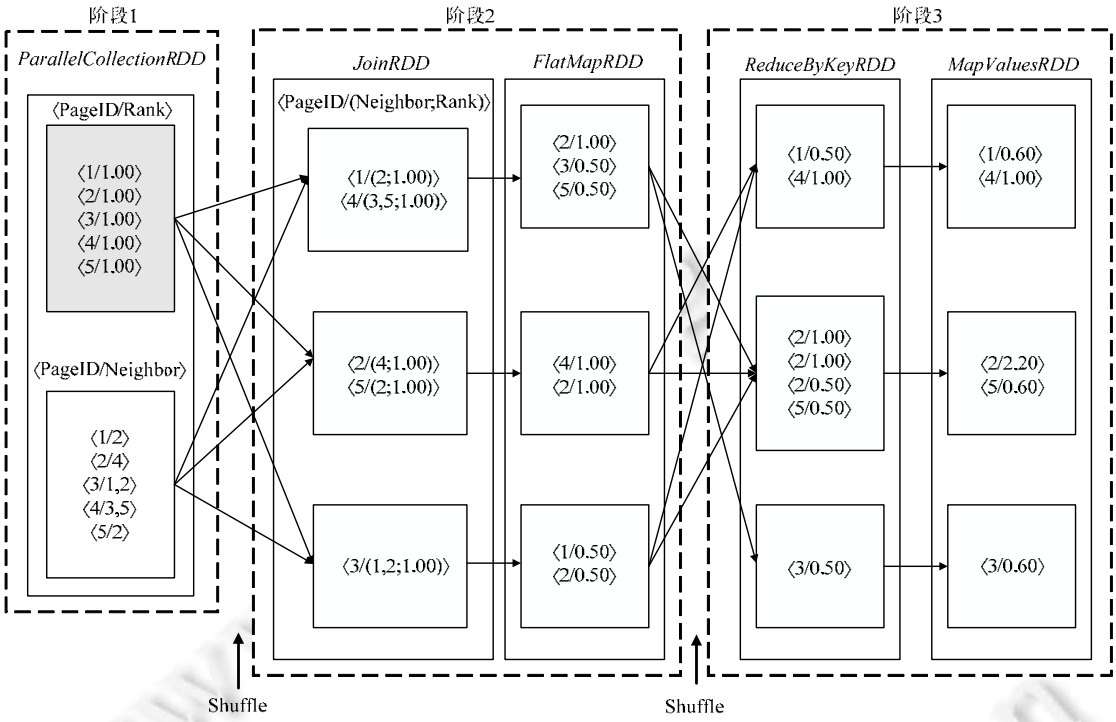

Fig.11 Spark for PageRank with one iteration

图 11 Spark 处理 PageRank 问题一次迭代

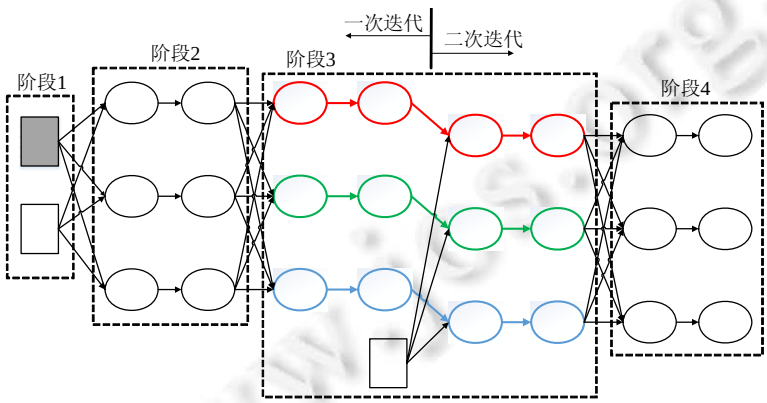

Fig.12 Spark for PageRank with two iterations

图 12 Spark 处理 PageRank 问题两次迭代

通过图 11 可以看到:在一次迭代过程中,Spark 需要 5 次转换(transformation)操作,共产生 5 个 RDD,与 MapReduce 一样需要进行两次 shuffle 操作,被划分为 3 个阶段.

与 MapReduce 不同的是,在图 12 的两次迭代中,Spark 可以将具有窄依赖关系的 RDD 分区分配到一个任务中进行管道化操作,任务内部数据无需通过网络传输且任务之间互不干扰.如在图 12 的阶段 3 中,所有分区节点根据依赖关系被分为 3 个任务(分别标注为红色、绿色和蓝色).在 PageRank 问题的两次迭代计算过程中,作业被划分为 4 个阶段,共有 11 个任务、3 次 shuffle 操作,Spark 相较于 MapReduce 有效地减少了任务之间的等

待时间以及中间数据传输数量.

5.4 PageRank问题分布式处理的比较与总结

与 WordCount 问题类似,我们将分别在一次迭代、两次迭代和 10 次迭代情况下,通过算法执行时间、文件数目和同步次数对 MapReduce 与 Spark 进行比较.

- 算法执行时间

首先讨论中间数据排序时间 T_{sort},从图 9 中可以看出:MapReduce 一次迭代需要两次 Map-Reduce 操作,第 1 次 Map-Reduce 操作中,每个 Map 都需要进行两次比较;而第 2 次 Map-Reduce 操作中,第 1 个 Map 任务进行的比较次数最多,为两次.则在一次迭代中,MapReduce 在中间数据排序上要进行 4 次比较,在两次迭代中,MapReduce,需要进行 8 次比较,在 10 次迭代中,MapReduce 需要进行 40 次比较.而 Spark 在 shuffle 时不要求中间数据有序,所以其比较次数为 0.

再讨论两者中间数据传输时间 T_{trans},在 WordCount 问题中,MapReduce 需要传输的中间数据量要小于 Spark;但在迭代问题中,MapReduce 的优势将不再明显.其原因是:Spark 根据依赖关系采用的任务调度策略,使得 shuffle 次数相较于 MapReduce 有了显著降低.

在 MapReduce 一次迭代过程中,需要进行两次数据传输,数据量分别为:$|D_1|=10$,$|D_2|=7$,$T_{trans\text{-}MR}=C_t\times 17$;在两次迭代中,MapReduce 需要进行 4 次数据传输,$|D_1|=10$,$|D_2|=7$,$|D_3|=10$,$|D_4|=7$,$T_{trans\text{-}MR}=C_t\times 34$;在 10 次迭代中,$T_{trans\text{-}MR}=C_t\times 170$.Spark 一次迭代过程中与 MapReduce 相同,需要进行两次传输:$|D_1|=10$,$|D_2|=7$,$T_{trans\text{-}Spark}=C_t\times 17$;在两次迭代中,需要进行 3 次数据传输,$|D_1|=10$,$|D_2|=7$,$|D_3|=7$,$T_{trans\text{-}Spark}=C_t\times 24$;而在 10 次迭代过程中,共进行了 11 次数据传输,$T_{trans\text{-}Spark}=C_t\times 80$.

- 文件数目

同 WordCount 类似,在需要进行网络传输的时候,MapReduce 的文件数量是 m,而 Spark 的文件数量是 $m\times r$,即,单次 shuffle 过程的文件数量 MapReduce 要少于 Spark.但从整个作业角度来看,Spark 的 shuffle 次数要少于 MapReduce,所以宏观上说,在迭代问题中,MapReduce 文件数量上的优势不如 WordCount 中那么明显.就此例而言,一次迭代过程中,MapReduce 总文件数为 2+3=5,Spark 总文件数为 6+9=15;两次迭代过程中,MapReduce 总文件数为 2+3+3+3=11,Spark 总文件数为 6+9+9=24;10 次迭代过程中,MapReduce 总文件数为 5+6×9=59,Spark 总文件数为 15+9×9=96.

- 同步次数

之前已经描述了同步模型与异步模型之间的区别,在算法执行过程中,同步次数越少、所占比例越小,越有利于算法的局部性能.虽然伴随着算法整体步骤的增加(如 Spark 中更多的遍历次数),但是总的来说会使算法性能得到有效的提升.在一次迭代过程中,MapReduce 与 Spark 同步次数皆为 2 次;在两次迭代过程中,MapReduce 中 4 次 Map-Reduce 操作全部遵守同步模型,同步次数为 4 次(即,每次都要等待上一步所有任务都执行完毕才会执行下一步操作),Spark 将两次迭代过程划分成 4 个阶段,每个阶段内遵守异步模型,阶段间为同步操作,所以同步次数为 3 次;在 10 次迭代过程中,MapReduce 共有 20 次同步操作,而 Spark 只有 11 次.

通过表 4 可以看到:在一次迭代过程中,MapReduce 与 Spark 在性能上并没有很大的差别;但是随着迭代次数的增加,两者的差距逐渐显现出来.

- 在算法执行时间方面,Spark 性能要好于 MapReduce.与 WordCount 问题不同,Spark 在 PageRank 问题中的中间数据传输数量要低于 MapReduce;
- 在同步次数方面,Spark 同步次数同样低于 MapReduce.

这些优势主要是因为 Spark 根据数据间的依赖关系对任务进行了更为合理的划分,有效地减少了问题中 shuffle 操作的次数.而因为 MapReduce 与 Spark 在 shuffle 操作中的差别,导致 Spark 文件数目要高于 MapReduce.而 Spark 也在不断地进行自我完善:在 Spark0.8.1 版本中引入了 shuffle consolidation[47]机制,有效地减少了 Spark 中间数据文件数量;从 Spark1.1 开始,Spark 开始将 MapReduce 的 Sort-BasedShuffle 机制作为一套备选 shuffle 机制供用户选择,Sort-BasedShuffle 机制能够保证每个 Map 任务节点只输出一个文件,且数据为分区内有序.

Table 4 Performance comparison between MapReduce and Spark with PageRank

表 4 MapReduce 与 Spark 处理 PageRank 问题之性能比较

		算法执行时间				文件数目	同步次数
		T_{read}	T_{write}	T_{trans}	T_{sort}		
一次迭代	MapReduce	相同	相同	C_t×17	4 次比较	5	2
	Spark			C_t×17	0 次比较	15	2
两次迭代	MapReduce	相同	相同	C_t×34	8 次比较	11	4
	Spark			C_t×24	0 次比较	24	3
10 次迭代	MapReduce	相同	相同	C_t×170	40 次比较	59	20
	Spark			C_t×80	0 次比较	96	11

6 总结与展望

本文首先分别阐述了 MapReduce 和 Spark 两种大数据分布式架构的背景、原理以及调度等方面,并从大数据评估指标和应用情景两个方面分别对二者进行对比和总结,针对不同问题需求给出相应的分析和选择.为了更好地理解 MapReduce 与 Spark 在原理上的区别,我们通过 WordCount 和 PageRank 算法分别分析 MapReduce 和 Spark 在处理非迭代和迭代问题时在算法执行时间、文件数目和同步次数上的差异.通过分析结果可以看出:在处理不同问题时,MapReduce 与 Spark 各有优点与不足.在处理 WordCount 问题时,MapReduce 在排序上消耗了更多的时间,而在中间数据传输、文件数目以及同步次数上效率都要优于 Spark;在处理 PageRank 问题时,Spark 通过更合理的任务调度机制,在算法执行时间和同步次数方面要优于 MapReduce.但是因为不同的 shuffle 机制,导致 Spark 的文件数目要高于 MapReduce,而 Spark 也在后期的版本中对此缺点进行了补充和完善.

MapReduce 作为第一代大数据分布式架构,让传统的大数据问题可以并行地在多台处理机上进行计算.而 MapReduce 之所以能够迅速成为大数据处理的主流计算平台,得力于其自动并行、自然伸缩、实现简单和容错性强等特性[48].但是 MapReduce 并不适合处理迭代问题以及低延时问题,而 Spark 作为轻量、基于内存计算的分布式计算平台,采用了与 MapReduce 类似的编程模型,使用 RDD 抽象对作业调度、数据操作和存取进行修改,并增加了更丰富的算子,使得 Spark 在处理迭代问题、交互问题以及低延时问题时能有更高的效率.同样,Spark 也有其不足:如数据规模过大或内存不足时,会出现性能降低、数据丢失需要进行重复计算等缺点.总而言之,随着大数据领域的不断发展和完善,现有的大数据分析技术仍然有大量具有挑战性的问题需要深入研究,而作为大数据领域重要的两种分布式处理架构,MapReduce 与 Spark 都有着不可替代的地位和作用.

References:

[1] Gordon K. What is big data? Itnow, 2013,55(3):12−13.

[2] Dean J, Ghemawat S. MapReduce: Simplified data processing on large clusters. In: Proc. of the 6th Conf. on Symp. on Opearting Systems Design and Implementation. San Francisco: USENIX Association Berkeley, 2004. 137−149.

[3] Rao BT, Sridevi N, Reddy VK, Reddy L. Performance issues of heterogeneous hadoop clusters in cloud computing. Global Journal of Computer Science and Technology, 2011,6(8):1−6.

[4] Zaharia M, Chowdhury M, Franklin MJ, Shenker S, Stoica I. Spark: Cluster computing with working sets. HotCloud, 2010,10 (10-10):1−7.

[5] Dean J. Experiences with MapReduce, an abstraction for large-scale computation. In: Proc. of the 15th Int'l Conf. on Parallel Architectures and Compilation Techniques. Washington: ACM Press, 2006. 1−1.

[6] Srirama SN, Jakovits P, Vainikko E. Adapting scientific computing problems to clouds using MapReduce. Future Generation Computer Systems, 2012,28(1):184−192. [doi: 10.1016/j.future.2011.05.025]

[7] Elghandour I, Aboulnaga A. ReStore: Reusing results of MapReduce jobs in pig. In: Proc. of the 2012 ACM SIGMOD Int'l Conf. on Management of Data. Scottsdale: ACM Press, 2012. 701−704. [doi: 10.1145/2213836.2213937]

[8] Pansare N, Borkar VR, Jermaine C, Condie T. Online aggregation for large MapReduce jobs. Proc. of the Vldb Endowment, 2012, 4(11):1135−1145.

[9] Zaharia M, Borthakur D, Sarma JS, Elmeleegy K, Shenker S, Stoica I. Job scheduling for multi-user MapReduce clusters. Technical Report UCB/EECS-2009-55. Berkeley: EECS Department, University of California, 2009. 1−18.

[10] Zaharia M. Job scheduling with the fair and capacity schedulers. In: Proc. of the Hadoop Summit. 2009. 9.

[11] Nightingale EB, Chen PM, Flinn J. Speculative execution in a distributed file system. ACM SIGOPS Operating Systems Review, 2005,39(5):191−205. [doi: 10.1145/1095809.1095829]

[12] Isard M, Budiu M, Yu Y, Birrell A, Fetterly D. Dryad: Distributed data-parallel programs from sequential building blocks. In: Proc. of the 2nd ACM SIGOPS/EuroSys European Conf. on Computer Systems 2007. Lisbon: ACM Press, 2007. 59−72. [doi: 10.1145/1272998.1273005]

[13] Venner J. Pro Hadoop. Berkeley: Apress, 2009. 1−440. [doi: 10.1007/978-1-4302-1943-9]

[14] Shvachko K, Kuang H, Radia S, Chansler R. The hadoop distributed file system. In: Proc. of the 2010 IEEE 26th Symp. on Mass Storage Systems and Technologies (MSST). Nevada: IEEE, 2010. 1−10. [doi: 10.1109/MSST.2010.5496972]

[15] Vavilapalli VK, Murthy AC, Douglas C, Agarwal S, Konar M, Evans R, Graves T, Lowe J, Shah H, Seth S. Apache hadoop yarn: Yet another resource negotiator. In: Proc. of the 4th Annual Symp. on Cloud Computing. ACM Press, 2013. 1−16. [doi: 10.1145/2523616.2523633]

[16] Wang J, Shang P, Yin J. DRAW: A New Data-gRouping-AWare Data Placement Scheme for Data Intensive Applications with Interest Locality. New York: Springer, 2014. 149−174. [doi: 10.1007/978-1-4939-1905-5]

[17] Amer A, Long DD, Burns RC. Group-Based management of distributed file caches. In: Proc. of the 2002 22nd Int'l Conf. on Distributed Computing Systems. Vienna: IEEE, 2002. 525−534. [doi: 10.1109/ICDCS.2002.1022302]

[18] Chen Q, Zhang D, Guo M, Deng Q, Guo S. Samr: A self-adaptive MapReduce scheduling algorithm in heterogeneous environment. In: Proc. of the 2010 IEEE 10th Int'l Conf. on Computer and Information Technology (CIT). Bradford: IEEE, 2010. 2736−2743. [doi: 10.1109/CIT.2010.458]

[19] Kwon Y, Balazinska M, Howe B, Rolia J. Skewtune: Mitigating skew in MapReduce applications. In: Proc. of the 2012 ACM SIGMOD Int'l Conf. on Management of Data. Arizona: ACM Press, 2012. 25−36. [doi: 10.1145/2213836.2213840]

[20] Cherniak A, Zaidi H, Zadorozhny V. Optimization strategies for A/B testing on HADOOP. Proc. of the VLDB Endowment, 2013, 6(11):973−984. [doi: 10.14778/2536222.2536224]

[21] Ekanayake J, Li H, Zhang B, Gunarathne T, Bae SH, Qiu J, Fox G. Twister: A runtime for iterative MapReduce. In: Proc. of the 19th ACM Int'l Symp. on High Performance Distributed Computing. Chicago: ACM Press, 2010. 810−818. [doi: 10.1145/1851476.1851593]

[22] Bu Y, Howe B, Balazinska M, Ernst MD. HaLoop: Efficient iterative data processing on large clusters. Proc. of the VLDB Endowment, 2010,3(1-2):285−296. [doi: 10.14778/1920841.1920881]

[23] Zhang Y, Gao Q, Gao L, Wang C. iMapReduce: A distributed computing framework for iterative computation. Journal of Grid Computing, 2012,10(1):47−68. [doi: 10.1007/s10723-012-9204-9]

[24] Zaharia M, Chowdhury M, Das T, Dave A, Ma J, McCauley M, Franklin MJ, Shenker S, Stoica I. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In: Proc. of the 9th USENIX Conf. on Networked Systems Design and Implementation. San Jose: USENIX Association Berkeley, 2012. 1−14.

[25] Yu Y, Isard M, Fetterly D, Budiu M, Erlingsson Ú, Gunda PK, Currey J. DryadLINQ: A system for general-purpose distributed data-parallel computing using a high-level language. In: Proc. of the 8th USENIX Conf. on Operating Systems Design and Implementation. USENIX Association Berkeley, 2008. 1−14.

[26] Lhoták O, Hendren L. Scaling Java points-to analysis using Spark. In: Proc. of the 12th Int'l Conf. on Compiler Construction. Berlin: Springer-Verlag, 2003. 153−169.

[27] Ananthanarayanan G, Ghodsi A, Shenker S, Stoica I. Disk-Locality in datacenter computing considered irrelevant. In: Proc. of the 13th USENIX Conf. on Hot topics in Operating Systems. California USENIX Association Berkeley, 2011. 12−17.

[28] Bhatotia P, Wieder A, Rodrigues R, Acar UA, Pasquin R. Incoop: MapReduce for incremental computations. In: Proc. of ACM Symp. on Cloud Computing. Cascais: ACM Press, 2011. 1−14. [doi: 10.1145/2038916.2038923]

[29] Armbrust M, Xin RS, Lian C, Huai Y, Liu D, Bradley JK, Meng X, Kaftan T, Franklin MJ, Ghodsi A. Spark sql: Relational data processing in Spark. In: Proc. of the 2015 ACM SIGMOD Int'l Conf. on Management of Data. Victoria: ACM Press, 2015. 1383−1394. [doi: 10.1145/2723372.2742797]

[30] Zaharia M, Das T, Li H, Hunter T, Shenker S, Stoica I. Discretized streams: Fault-tolerant streaming computation at scale. In: Proc. of the 24th ACM Symp. on Operating Systems Principles. Farmington: ACM Press, 2013. 423−438. [doi: 10.1145/2517349.2522737]

[31] Meng X, Bradley J, Yavuz B, Sparks E, Venkataraman S, Liu D, Freeman J, Tsai D, Amde M, Owen S, Xin D, Reynold X, Franklin MJ, Zadeh R, Zaharia M, Talwalkar A. Mllib: Machine learning in apache Spark. The Journal of Machine Learning Research, 2016, 17(1):1235−1241.

[32] Xin RS, Gonzalez JE, Franklin MJ, Stoica I. Graphx: A resilient distributed graph system on Spark. In: Proc. of the 1st Int'l Workshop on Graph Data Management Experiences and Systems. New York: ACM Press, 2013. 1−6. [doi: 10.1145/2484425.2484427]

[33] Gonzalez JE, Xin RS, Dave A, Crankshaw D, Franklin MJ, Stoica I. GraphX: Graph processing in a distributed dataflow framework. In: Proc. of the 11th USENIX Conf. on Operating Systems Design and Implementation. USENIX Association Berkeley, 2014. 599−613.

[34] Malewicz G, Austern MH, Bik AJ, Dehnert JC, Horn I, Leiser N, Czajkowski G. Pregel: A system for large-scale graph processing. In: Proc. of the 2010 ACM SIGMOD Int'l Conf. on Management of Data. ACM Press, 2010. 135−146. [doi: 10.1145/1807167.1807184]

[35] Low Y, Bickson D, Gonzalez J, Guestrin C, Kyrola A, Hellerstein JM. Distributed GraphLab: A framework for machine learning and data mining in the cloud. Proc. of the VLDB Endowment, 2012,5(8):716−727. [doi: 10.14778/2212351.2212354]

[36] Thomas K, Grier C, Ma J, Paxson V, Song D. Design and evaluation of a real-time URL spam filtering service. In: Proc. of the 2011 IEEE Symp. on Security and Privacy (SP). Berkeley: IEEE, 2011. 447−462. [doi: 10.1109/SP.2011.25]

[37] Zhang Z, Barbary K, Nothaft FA, Sparks E, Zahn O, Franklin MJ, Patterson DA, Perlmutter S. Scientific computing meets big data technology: An astronomy use case. In: Proc. of the 2015 IEEE Int'l Conf. on Big Data. Santa Clara: IEEE, 2015. 918−927. [doi: 10.1109/BigData.2015.7363840]

[38] Nothaft FA, Massie M, Danford T, Zhang Z, Laserson U, Yeksigian C, Kottalam J, Ahuja A, Hammerbacher J, Linderman M. Rethinking data-intensive science using scalable analytics systems. In: Proc. of the 2015 ACM SIGMOD Int'l Conf. on Management of Data. Victoria: ACM Press, 2015. 631−646. [doi: 10.1145/2723372.2742787]

[39] Veiga J, Expósito RR, Pardo XC, Taboada GL, Tourifio J. Performance evaluation of big data frameworks for large-scale data analytics. In: Proc. of the 2016 IEEE Int'l Conf. on Big Data. Washington: IEEE, 2016. 424−431. [doi: 10.1109/BigData. 2016.7840633]

[40] Lee H, Kang M, Youn SB, Lee JG, Kwon Y. An experimental comparison of iterative MapReduce frameworks. In: Proc. of the 25th ACM Int'l on Conf. on Information and Knowledge Management. Indiana: ACM Press, 2016. 2089−2094. [doi: 10.1145/2983323.2983647]

[41] Gu L, Li H. Memory or time: Performance evaluation for iterative operation on hadoop and Spark. In: Proc. of the 2013 IEEE 10th Int'l Conf. on High Performance Computing and Communications & 2013 IEEE Int'l Conf. on Embedded and Ubiquitous Computing (HPCC_EUC). Zhangjiajie: IEEE, 2013. 721−727. [doi: 10.1109/HPCC.and.EUC.2013.106]

[42] Kang M, Lee JG. An experimental analysis of limitations of MapReduce for iterative algorithms on Spark. Cluster Computing, 2017,20(4):3593−3604. [doi: 10.1007/s10586-017-1167-y]

[43] Mavridis I, Karatza H. Performance evaluation of cloud-based log file analysis with apache hadoop and apache Spark. Journal of Systems and Software, 2017,125:133−151. [doi: 10.1016/j.jss.2016.11.037]

[44] Song J, Sun ZZ, Mao KM, Bao YB, Yu G. Research advance on MapReduce based big data processing platforms and algorithms. Ruan Jian Xue Bao/Journal of Software, 2017,28(3):514−543 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5169.htm

[45] Shi J, Qiu Y, Minhas UF, Jiao L, Wang C, Reinwald B, Özcan F. Clash of the titans: Mapreduce vs. Spark for large scale data analytics. Proc. of the VLDB Endowment, 2015,8(13):2110−2121. [doi: 10.14778/2831360.2831365]

[46] Page L. The PageRank citation ranking: Bringing order to the Web. Stanford Digital Libraries Working Paper, 1998,9(1):1−17.

[47] Davidson A, Or A. Optimizing shuffle performance in Spark. Technical Report, University of California, Berkeley-Department of Electrical Engineering and Computer Sciences, 2013.

[48] Wolf J, Balmin A, Rajan D, Hildrum K, Khandekar R, Parekh S, Wu KL, Vernica R. On the optimization of schedules for MapReduce workloads in the presence of shared scans. The Int'l Journal on Very Large Data Bases, 2012,21(5):589−609. [doi: 10.1007/s00778-012-0279-5]

附中文参考文献:

[44] 宋杰,孙宗哲,毛克明,鲍玉斌,于戈.MapReduce 大数据处理平台与算法研究进展.软件学报,2017,28(3):514−543. http://www.jos.org.cn/1000-9825/5169.htm [doi: 10.13328/j.cnki.jos.005169]

吴信东(1963－),男,安徽枞阳人,博士,教授,博士生导师,主要研究领域为数据挖掘,大数据分析,知识库系统,万维网信息探索.

嵇圣硙(1994－),男,硕士生,主要研究领域为数据挖掘,分布式数据库.