文献综述

计算机学院　硕 1709　M201773112　龚孙璐

1. 背景

大型数据集的分析是一个重要的问题，目前市面上已经有很多大型数据系统如 Apache 公司出品的 Giraph[1]、基于 Hadoop 环境的 Impala[2]，在分布式条件下具有比较好的并行度的 GraphLab[3]、能执行循环数据流程序和并行处理数据的分布式系统 Naiad[4]、以及本文中主要的实验环境 Myria[5]等都可以帮助完成这个难题。上述的这些系统都是基于 shared-nothing 架构的集群，共享功能是通过许多请求或者称为应用来实现的，而且同一时刻同一个集群上可能运行着多个系统。在这样一种架构的集群内，系统和应用之间的资源分配是通过如 Mesos[6]或者 YARN[7]等的资源管理器来进行管理的。现代资源管理器是基于如 YARN[7]、Docker[8]或是 Kubernetes[9]等容器技术实现的，因为容器能够将共享同一机器的应用互相隔离开，且能提供硬件资源限制，这使得应用所需资源能被容器所限制和保护。简单来说，就是资源管理器控制容器的启动并且限制管理其可以使用的资源限制，然后分配应用到容器中运行。

许多现代数据分析系统都力图实现最大的内存利用率因为内存目前仍是比较重要且昂贵的资源，然而，基于容器的内存调度依旧有诸多限制，比如当一个应用准备运行时，会估算它的资源需求并且将这些发送给资源管理器，然后资源管理器根据可用资源的数量来决定是否要调度运行该应用，但是问题是，一个数据分析应用的内存需求在其运行前很难估算，因为它受多个包括中间结果的基数等执行期的因素影响，这些是很难估算的[10][11]。

而不准确的内存使用的预测会从多方面降低查询性能，例如如果估计得太高，集群的资源的利用率会很低，但是如果估计值低于完成一个查询所需内存的最小值，那么系统有可能会发生数据溢出，从而导致性能退化，或者产生 out-of-memory 的错误而查询失败，浪费查询已经使用的内存。这些问题在手动内存管理系统中都出现过，比如用 C/C++写的系统[2][3]、基于 Java 的字节数组系统[12]就存在这种问题，且在基于 Java 库的[1][5] [13][14]和.NET 公共语言库(CLR)的[4]自动内存管理系统中也出现过。而在自动内存分配系统中，当使用垃圾收集功能（GC）时情况将变得更复杂，因为 GC 会给查询性能增添另一层的不可预料性。甚至就算预测的资源足够完成查询请求，GC 在有些情况下也会显著的降低查询性能。

弹性内存管理有望解决这些问题，但是弹性容器的内存管理是一个很难的问题。首先，大多数系统都不支持弹性内存分配。在基于 Java 的系统中，Java 虚拟机（JVM）的最大堆大小在其整个生命周期中是固定的。而在基于 C/C++的系统如 Impala[2]中，限制一个进程的资源通常都是用 Linux 应用如 cgroups 实现的，而这种应用通常都不提供在活动期改变资源限制的对外端口。对于运行在 CLR 上的系统[4]，情况就完全相反：在这些系统上并没有对于堆大小的控制，所以堆大小可以随着总的物理内存的增长而自由增长。其次，为了能弹性地和动态的分配内存给数据分析应用，我们需要弄清楚提供多少额外的内存可以防止避免查询失败和加速应用，并且需要建立一个 GC 的效益和开销的模型。最后，需要一个算法去使得这个模型能处理好多个数据分析应用的内存分配的策划问题。

因此，本文设计了一个名为 ElasticMem 的解决方案，通过一定的策略将内存弹性的分配给运行有数据分析应用的相互隔离的容器，实现在总内存一定的限制下的查询失败最少以及运行时间的最优。

ElasticMem 的主要创新点在于：动态 JVM 堆大小的实现、动态内存的分配和优化，运行参数估算模型。

2. 弹性内存分配法

2.1 弹性调节 JVM 堆大小的配置。

普通的 OpenJDK 的应用管理是：首先，用户在应用启动前自定自己的最大堆大小，JVM 询问系统后，通过其内部的分配策略将内存分配给用户，在用户程序运行期间，如果提前分配给用户的内存不够用，则 JVM 就会启动 GC 程序回收内存，然后分配给该用户，如果没有足够的内存能回收，那么就发出一个 OutOfMemory 的错误。在 JVM 的整个运行期间，最大堆大小都是不变的，这就导致，当出现 OutOFMemory 错误的时候，即使宿主机上还有多余的内存空间可用，JVM 也不会提高最大堆大小，同样的，当宿主机的内存使用紧缩时，也不会降低最大堆大小。

这种设计现在看来是很没有必要的，因为现代的计算机操作系统都支持超量内存使用技术，通过 64 位的内存地址空间，可以在启动一个 JVM 时提供更多的逻辑内存地址，而逻辑内存在其被使用前不会实际的占用内存。

所以在这个技术的基础上，该研究对 OpenJDK 的源代码做了一些改动。首先让 JVM 在启动的时候保留一段连续地址空间，每代应用的初始堆大小限制还是按照 JVM 的内部策略进行，这样可以将每代应用的最大堆大小设置得足够大以保持其互相之间的独立性。然后，对每个 JVM 的新应用和旧应用都设置一个动态的大小限制，并将其设定成合适大小，如 1GB。接着，还设计一组基于套接字的 API 来实现交互。此外，还禁用了 JVM 内置的 GC 策略，替换成指定的策略，即让 GC 一直运行来回收内存，当现有空闲内存不能满足请求的更多内存时，挂起 JVM 直到足够多的内存被回收回来。

2.2 动态内存分配

本文中提及的 ElasticMem 系统的主要部分就是内存管理。在本文中，研究者将 JVM 中的每个操作都赋予一个权值，一个权值受多方面因素的影响如这个操作是否关闭了一个 JVM，是否导致 JVM 的挂起，或者这个操作使 JVM 能以多大效率获取内存。

在给每个操作定义权值后，资源管理器会根据两个方面来进行决策，一个是 JVM 的堆状态信息，一个是运行在 JVM 上的操作的权值的估计值。对于操作的权值，简单来说，计算按不同的操作的所用的时间和分配的内存的大小的比值作为性价比值，其中 KILL 和 NOOP 操作的值最高为 1，然后，计算每个 JVM 上所有操作的性价比值的和，找出其中 KILL 操作和 NOOP 操作最少且值最低的 JVM，给这个 JVM 赋予较高的权值，剩余的 JVM 也按如此方法设置权值，就得到了一个最优的调度策略。

2.3 运行参数估计

ElasticMem 系统的最后一部分是参数估计模型。首先是堆的增长估计，通过让管理器每 b 个步长的时间间隔内记录下新代应用内存使用的最大变化，将其作为堆的增长的估计值。

接着是 CG 所用的时间和节省的内存的估计。具体的 GC 时间和节省内存与收集域的所有活动或死亡的对象的大小和数量来决定的，事实上要获取这些数据要花费很大的代价，这种代价甚至比原本要减少的 GC 开销还要大，所以这种方法并不可行。所以，本文打算从请求操作的数据结构入手，因为这些数据结构与对象的状态相关，本文将监控请求操作的主要数据结构，然后收集统计数据作为其特征从而建立模型。在大型的数据分析系统中会有很多大型的数据结构，不过查询操作所涉及到的基本上都是哈希表的结构，所以模型是基序哈希表这种数据类型来构建的。大致思路就是先建立一个针对一个哈希表的模型，然后收集每个

哈希表中的模型得到的统计数据，并将其作为整个查询的预测值。

3. 评测

3.1 调度评测

所有的评测都是基于 Myria 上运行的，通过运行 TPC-H 标准请求量化评测其性能，其中，为了让 TPC-H 语句能在 Myria 下运行，用 Myria 的语言 MyriaL 对其进行编写。

在连续查询调度测评中，实验结果表明，该动态资源管理系统有如下优点：它能够自动调整并行度且在避免 out-of-memory 错误的同时能使系统实现最好的性能。并且在单位内存增量不同的情况下，对于原生的资源管理器在查询处理上有 10%到 30%的性能提升，在 GC 操作上有 40%到 80%的提升。但是当内存使用比较紧张时，弹性调度系统由于要尽可能实现多查询并行，所以在性能上会逊色于原生系统。

在延迟查询调度测评中，每个查询时间都会设置一个时延 30s，实验结果和连续查询类似，不同的是，当内存使用比较紧张时，弹性系统依旧可以在更少的时间内处理相同数量的查询，在查询完成度和使用时间上完胜原生系统。

至于时间步间隔的影响，实验结果表明弹性资源管理系统对时间步长的变化并不敏感，所以不用在这方面对其进行精确的调整。

3.2 GC 模型

本文选择 Weka 中的 M5P 模型进行测试，M5P 是一个叶子节点是线性回归的决策树。对训练集和 TPC-H 随机测试序列进行 10-折交叉验证，实验结果表明，总的来说，预测错误率还算可以接受，并且预测结果已经足够帮助资源管理器做出一个较好的内存分配决策。

总的来说，相对于 JVM 原生的资源调度方案，本文所提及的方法能够在查询处理和 GC 上有很显著的性能提升。

4. 相关工作

单个物理机内的内存分配：目前有很多方法都聚焦于单机上多个对象的内存共享，有几个将请求作为对象的，有些是基于页面访问模型来分配缓存空间从而减少分页错误[15][16][17]，还有一些通过调节缓存分配策略从而满足实时数据库系统中的性能要求[18][19]，另一些还有利用应用资源的敏感度引导分配[20]。最近，Narasayya 等人设计了一种跨多租户的缓冲池分配算法[21]。同样有一些方法是基于将请求中的操作作为对象的，如 Anciaux 等设计了种在内存紧张设备中的跨操作的内存分配方法[22]，Davision 等通过将资源分配竞价分配给竞争性的操作，使得达到最大性价比[23]，Garofalakis 等的算法在 NUMA 系统中按多维的资源限制调度操作[24]，Storm 等的算法则是聚焦于跨数据库系统不同组成之间的内存管理[25]。尽管他们的算法都在不同的对象用一个对象函数实现了内存管理，不过由于在单机环境下，这些算法都具有其局限性，且没有考虑 GC 相关的问题。虽然 Salomie[26]等通过在 OpenJDK 中添加一个气球型空间实现了跨 JVM 的内存动态分配，不过并没有具体的性能模型和调度算法，所以并没办法量化评估其性能。Ginkgo[27]利用 Java 原生接口改变 Java 应用的布局从而实现了多应用间的动态内存分配，然而，他的性能模型是在特殊工作负载下评估的，而本文的方法在任意的关系查询中都适用。

集群尺度的资源调度：有些研究试图建立一个应用资源影响运行状态参数的模型，如 Li 等人按系统标定和优化器统计数据把不同类型机器上的请求分类[28]，Herodotou 等基于最大工作效益的机器学习模型调整 Hadoop 应用的参数的算法[29][30]。其他有一些研究则聚焦于短期请求，如 Lang 等的多租户事务型负载特殊硬件资源调度算法[31]，Schaffner 等的基于列数

据库集群的租户响应时间尾延迟最小算法[32]。不同于以上诸方法，本文则集中解决基于 Java 系统的非采样关系查询的相关问题，虽然已有一些资源管理系统提供了资源共享和应用调度的统一的框架[6][7][33]，但是这些系统都不能做到动态的调节内存限制。

适应性 GC 调节：Cook 等提出了两个基于实时统计量的 GC 触发策略，但是这个策略并没办法跨应用实现内存管理[34]，Simo 等在多个不同标准下评估了 JVM 堆大小增长对性能的影响[35]，Maas 等发现 GC 之间的协调对分布式应用相当重要[36]，他们都是让用户指定协调策略以便所有的 JVM 都在某个条件下的同一时刻启动 GC。

基于区域的内存管理：另外一个思路是通过基于区域的内存管理（RBMM）[37]来避免 GC 过载，如 Broom 给 Naiad 的每个对象标上一个区域标签，然后根据区域将其分为三种类型[38]，虽然 RBMM 可能会减少 GC 过载，但是它要求程序员声明对象到区域间的映射，这增加了编译的复杂度，并且忽略了内存空间的安全性问题。

5.结论

本文提出了一个基于无共享架构集群的大型数据分析应用的自动单行内存管理方法 ElasticMem，该方法主要通过 JVM 内存限制的动态调整，运行时的内存使用和 GC 代价模型以及减少总的错误数和运行时间的内存管理实现。通过在 Myria 系统上的评测证实该方法无论是在查询失败还是运行时间上都胜过静态内存管理系统。

参考文献：

[1] Apache Giraph. http://giraph.apache.org/.

[2] M. Kornacker, A. Behm, V. Bittorf, T. Bobrovytsky, C. Ching, A. Choi, J. Erickson, M. Grund, D. Hecht, M. Jacobs, I. Joshi, L. Kuff, D. Kumar, A. Leblang, N. Li, I. Pandis, H. Robinson, D. Rorke, S. Rus, J. Russell, D. Tsirogiannis, S. Wanderman-Milne, and M. Yoder. Impala: A modern, open-source SQL engine for Hadoop. In *CIDR*, 2015.

[3] Y. Low, D. Bickson, J. Gonzalez, C. Guestrin, A. Kyrola, and J. M. Hellerstein. Distributed graphlab: A framework for machine learning and data mining in the cloud. *Proc*. VLDB *Endow*., 5(8):716–727, Apr. 2012.

[4] D. G. Murray, F. McSherry, R. Isaacs, M. Isard, P. Barham, and M. Abadi. Naiad: A timely dataflow system. In P*roceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, SOSP ’13, pages 439–455, New York, NY, USA, 2013. ACM.

[5] J. Wang, T. Baker, M. Balazinska, D. Halperin, B. Haynes, B. Howe, D. Hutchison, S. Jain, R. Maas, P. Mehta, D. Moritz, B. Myers, J. Ortiz, D. Suciu, A. Whitaker, and S. Xu. The Myria big data management and analytics system and cloud services. In *CIDR*, 2017.

[6] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. Katz, S. Shenker, and I. Stoica. Mesos: A platform for fine-grained resource sharing in the data center. In *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation*, NSDI’11, pages 295– 308, Berkeley, CA, USA, 2011. USENIX Association.

[7] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth, B. Saha, C. Curino, O. O’Malley, S. Radia, B. Reed, and E. Baldeschwieler. Apache Hadoop YARN: Yet another resource negotiator. In P*roceedings of the 4th Annual Symposium on Cloud Computing*, SOCC ’13, pages 5:1– 5:16, New York, NY, USA, 2013. ACM.

[8] Docker Container. https://www.docker.com/.

[9] Kubernetes. http://kubernetes.io/.

[10] Y. E. Ioannidis and S. Christodoulakis. On the propagation of errors in the size of join results. In *Proceedings of the 1991 ACM SIGMOD International Conference on Management of Data*, SIGMOD ’91, pages 268–277, New York, NY, USA, 1991. ACM.

[11] V. Leis, A. Gubichev, A. Mirchev, P. Boncz, A. Kemper, and T. Neumann. How good are query optimizers, really? *Proc*. VLDB *Endow*., 9(3):204–215, Nov. 2015.

[12] Tungsten: memory management and binary processing on Spark. https://databricks.com/blog/2015/04/28/project-tungsten-bringing-spark-closer-tobare-metal.html.

[13] T. White. *Hadoop: The Definitive Guide*. O’Reilly Media, Inc., 1st edition, 2009.

[14] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. *In Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, NSDI’12, pages 2–2, Berkeley, CA,

[15] C. M. Chen and N. Roussopoulos. Adaptive database buffer allocation using query feedback. In *Proceedings of the 19th International Conference on Very Large Data Bases*, VLDB ’93, pages 342–353, San Francisco, CA, USA, 1993. Morgan Kaufmann Publishers Inc.

[16] C. Faloutsos, R. T. Ng, and T. K. Sellis. Predictive load control for flexible buffer allocation. In *Proceedings of the 17th International Conference on Very Large Data Bases*, VLDB ’91, pages 265–274, San Francisco, CA, USA, 1991. Morgan Kaufmann Publishers Inc.

[17] R. Ng, C. Faloutsos, and T. Sellis. Flexible buffer allocation based on marginal gains. In P*roceedings of the 1991 ACM SIGMOD International Conference on Management of Data*, SIGMOD ’91, pages 387–396, New York, NY, USA, 1991. ACM.

[18] K. P. Brown, M. J. Carey, and M. Livny. Managing memory to meet multiclass workload response time goals. In *Proceedings of the 19th International Conference on Very Large Data Bases*, VLDB ’93, pages 328–341, San Francisco, CA, USA, 1993. Morgan Kaufmann Publishers Inc.

[19] H. H. Pang, M. J. Carey, and M. Livny. Managing memory for real-time queries. In *Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data*,

SIGMOD '94, pages 221–232, New York, NY, USA, 1994. ACM.
[20] P. Tembey, A. Gavrilovska, and K. Schwan. Merlin: Application- and platform-aware resource allocation in consolidated server systems. In *Proceedings of the ACM Symposium on Cloud Computing*, SOCC '14, pages 14:1– 14:14, New York, NY, USA, 2014. ACM.
[21] V. Narasayya, I. Menache, M. Singh, F. Li, M. Syamala, and S. Chaudhuri. Sharing buffer pool memory in multitenant relational database-as-a-service. *Proc*. VLDB *Endow*., 8(7):726–737, Feb. 2015.
[22] N. Anciaux, L. Bouganim, and P. Pucheral. Memory requirements for query execution in highly constrained devices. In *Proceedings of the 29th International Conference on Very Large Data Bases.* Volume 29, VLDB '03, pages 694–705. VLDB Endowment, 2003.
[23] D. L. Davison and G. Graefe. Dynamic resource brokering for multi-user query execution. In *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data*, SIGMOD '95, pages 281–292, New York, NY, USA, 1995. ACM.
[24] M. N. Garofalakis and Y. E. Ioannidis. Parallel query scheduling and optimization with time- and space-shared resources. In *Proceedings of the 23rd International Conference on Very Large Data Bases*, VLDB '97, pages 296– 305, San Francisco, CA, USA, 1997. Morgan Kaufmann Publishers Inc.
[25] A. J. Storm, C. Garcia-Arellano, S. S. Lightstone, Y. Diao, and M. Surendra. Adaptive self-tuning memory in db2. In *Proceedings of the 32Nd International Conference on Very Large Data Bases*, VLDB '06, pages 1081–1092. VLDB Endowment, 2006.
[26] T. I. Salomie, G. Alonso, T. Roscoe, and K. Elphinstone. Application level ballooning for efficient server consolidation. In *Proceedings of the 8th ACM European Conference on Computer Systems*, EuroSys '13, pages 337–350, New York, NY, USA, 2013. ACM.
[27] M. R. Hines, A. Gordon, M. Silva, D. Da Silva, K. Ryu, and M. Ben-Yehuda. Applications know best: Performance-driven memory overcommit with ginkgo. In *Proceedings of the 2011 IEEE Third International Conference on Cloud Computing Technology and Science*, CLOUDCOM '11, pages 130–137,Washington, DC, USA, 2011. IEEE Computer Society.
[28] J. Li, J. Naughton, and R. V. Nehme. Resource bricolage for parallel database systems. *Proc*. VLDB *Endow*., 8(1):25–36, Sept. 2014.
[29] H. Herodotou, F. Dong, and S. Babu. No one (cluster) size fits all: Automatic cluster sizing for data-intensive analytics. In *Proceedings of the 2nd ACM Symposium on Cloud Computing*, SOCC '11, pages 18:1–18:14, New York, NY, USA, 2011. ACM.
[30] H. Herodotou, H. Lim, G. Luo, N. Borisov, L. Dong, F. B. Cetin, and S. Babu. Starfish: A self-tuning system for big data analytics. In *In CIDR*, pages 261–272, 2011.
[31] W. Lang, S. Shankar, J. M. Patel, and A. Kalhan. Towards multi-tenant performance slos. *IEEE Trans. on Knowl. and Data Eng*., 26(6):1447–1463, June 2014.
[32] J. Schaffner, B. Eckart, D. Jacobs, C. Schwarz, H. Plattner, and A. Zeier. Predicting in-memory database performance for automating cluster management tasks. In *ICDE*, pages 1264–1275. IEEE Computer Society, 2011.
[33] M. Weimer, Y. Chen, B.-G. Chun, T. Condie, C. Curino, C. Douglas, Y. Lee, T. Majestro, D. Malkhi, S. Matusevych, B. Myers, S. Narayanamurthy, R. Ramakrishnan, S. Rao, R. Sears, B. Sezgin, and J. Wang. Reef: Retainable evaluator execution framework. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, SIGMOD '15, pages 1343–1355, New York, NY, USA, 2015. ACM.
[34] J. E. Cook, A. W. Klauser, A. L. Wolf, and B. G. Zorn. Semi-automatic, self-adaptive control of garbage collection rates in object databases. In P*roceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, SIGMOD '96, pages 377–388, New York, NY, USA, 1996. ACM.
[35] J. Sim˜ao and L. Veiga. Adaptability driven by quality of execution in high level virtual machines for shared cloud environments. *Comput. Syst. Sci. Eng*., 28(6), 2013.
[36] M. Maas, T. Harris, K. Asanovic, and J. Kubiatowicz. Trash day: Coordinating garbage collection in distributed systems. In *15th Workshop on Hot Topics in Operating Systems, HotOS XV, Kartause Ittingen, Switzerland, May 18-20, 2015*, 2015.
[37] M. Tofte and J.-P. Talpin. Region-based memory management. *Inf. Comput*., 132(2):109–176, Feb. 1997.

[38] I. Gog, J. Giceva, M. Schwarzkopf, K. Vaswani, D. Vytiniotis, G. Ramalingam, M. Costa, D. G. Murray, S. Hand, and M. Isard. Broom: Sweeping out garbage collection from big data systems. In *15th Workshop on Hot Topics in Operating Systems (HotOS XV), Kartause Ittingen, Switzerland, 2015*. USENIX Association.