文章编号:2096-1472(2023)-07-44-04
DOI:10.19644/j.cnki.issn2096-1472.2023.007.010

一种优化的 Hadoop 数据放置策略

吴 岳

(国家林业和草原局产业发展规划院, 北京 100010)
✉ wuyue98@126.com

摘 要:Hadoop 分布式文件系统(HDFS)的默认数据块放置策略均衡了数据存储的可靠性和读写速度,却没有考虑发挥集群的最佳性能。针对该问题提出了一种优化后的数据块放置算法。该算法为数据块设计 2 个指标,即被查询率与平均读取时间,用于评估集群执行任务对数据块的需求量。在符合 HDFS 默认数据放置算法基本规则的前提下,通过对数据块的需求量进行分析,然后重新计算数据块的放置位置,将需求量最多的数据转移到能够最快处理它们的节点上。通过实验数据证明:该算法可以使集群整体性能提高 20%以上。优化后的数据块放置算法是有效的,并且不会增加对集群带宽的占用。

关键词:HDFS;数据块;放置策略;性能优化
中图分类号:TP311.1 **文献标识码**:A

An Optimized Hadoop Data Placement Strategy

WU Yue

(*State Forestry and Grassland Administration Industrial Development Planning Institute, Beijing* 100010, *China*)
✉ wuyue98@126.com

Abstract: The default data chunk placement strategy of Hadoop Distributed File System (HDFS) balances the reliability of data storage and read/write speed, but does not consider the optimal performance of the cluster. The paper proposes an optimized data placement algorithm to address this issue. Two indicators for data chunks, namely query rate and average read time are designed in this algorithm, to evaluate the demand of data chunks for cluster execution tasks. On the premise of meeting the basic rules of HDFS default placement algorithm, the data with the highest demand are transferred to the node that can process them the fastest, by analyzing the demand of chunks and recalculating their placement. Experimental data show that the algorithm can improve the overall performance of the cluster by more than 20%. The optimized data chunk placement algorithm is effective and will not increase the utilization of cluster bandwidth.

Keywords: HDFS; data chunks; placement strategy; performance optimization

0 引言(Introduction)

大数据系统的作用是存储和分析大量的数据,同时确保数据的安全性和可访问性。大数据系统将数据存储管理委托给分布式文件系统(DFS)执行,例如 Apache Hadoop 的数据存储基于 Hadoop 分布式文件系统(HDFS)[1]。Hadoop 可以实现扩展集群上的复杂计算,它的工作原理是将复杂的计算分布在多台机器上,并且使计算靠近数据,而不是将数据送至计算,最大限度地减少对网络带宽的占用,从而提高全局性能[2]。

HDFS 的数据放置策略会将数据块分条和复制,可在发生故障时尽可能地保护数据。该策略的数据保护原则是为相同数据创建多个副本,并将副本分布放置在多个机架的多台计算机上。然而,该策略放置副本时没有考虑到随着时间变化,每

个数据的被需求程度也会变化,有可能导致没有发挥集群的最优性能。本文通过深入研究 Hadoop 和 HDFS 的整体工作机制,提出一种综合考虑需求和处理性能实现重新放置副本的算法,并通过实验证明,该算法可以将集群的处理性能提高 20% 以上。

1 Hadoop 分布式文件系统(Hadoop Distributed File System)

1.1 HDFS 的架构

HDFS 的架构由单个活动的 Master(NameNode)和多个 Slave(DataNode)组成[3]。NameNode 负责管理集群中 DataNode 的状态,并处理整个集群中的存储分布。用户的每次读写操作都从向 NameNode 发起请求开始[4]。文件系统元信息(包含数据块定位)与数据块分别存储在 NameNode 和 DataNode 上,NameNode 只是将用户查询的 DataNode 列表作为元数据块传输给用户,之后用户在不重复读取 NameNode 的情况下与 DataNode 通信。NameNode 不参与用户和 DataNode 之间的数据传输,这种方法简化了 HDFS 的架构,避免了 NameNode 成为瓶颈[5]。

1.2 HDFS 的复制

HDFS 在复制过程中,将文件分成几个大小相等的数据块,这个操作称为数据条带化[6]。HDFS 通过将这些数据块存储在不同 DataNode 上实现并行化数据访问,从而缩短响应时间。将每个数据块复制成多个副本(默认是 3 个,可修改配置),并将每个副本存放在根据预定义策略确定的 DataNode 上[7]。这样可以降低由于硬件发生故障导致数据丢失的风险。DataNode 能通过心跳机制向 NameNode 汇报状态信息,当某个 DataNode 发生故障时,NameNode 可在其他 DataNode 上重构该 DataNode 上的数据块,保持每个数据块的副本数量正常[8]。

副本的数量也称为复制因子,由 NameNode 管理,可以随时针对每个文件进行单独更改[9]。创建文件时,NameNode 会向用户提供可以存储该数据的 DataNode 列表,该列表包含 N 个 DataNode,其中 N 是复制因子。用户先将数据块传输到第一个本地的 DataNode,之后该 DataNode 将数据块副本传输到列表中的第二个 DataNode,第二个 DataNode 继续传输副本,直到第 N 个 DataNode,数据就像在一个管道中传递。

1.3 HDFS 默认的副本放置策略

HDFS 通常被配置为由多个机架组成的集群,每个机架装配有多个 DataNode。不同机架之间的节点通信需要依靠交换机等网络设备,所以同一机架节点之间的数据交换速度会快于不同机架中的节点。如果只考虑网络带宽的占用和数据访问时间,可以从同一个机架上选择几个 DataNode 存储所有数据块的副本,但是这样的系统是脆弱的,如果该机架出现故障,则数据将会全部丢失。

HDFS 在执行数据块写入操作时,应用了一种平衡故障风险和访问速度的策略。如果复制因子为 3,第一个副本放置在与客户端相同节点的 DataNode 上,第二个副本放置在同一个机架的另一个 DataNode 上,第三个副本放置在不同机架的 DataNode 上。如果复制因子大于 3,第四个及后续的副本都随机放置在集群中。这种策略可以在网络、机架或交换机出现故障时提供更好的数据健壮性,并且优化了写入时间,因为 2/3 的数据量是在同一机架间交换的(复制因子为 3 时)。

这种策略存在一个问题,它在确定放置副本的 DataNode 时,没有考虑对副本的访问需求,事实上在创建数据块时,第一个副本总是放置在靠近"写入者"的 DataNode 上。这种副本放置策略可能会因集群上数据分布不均而影响处理性能。本文提出一种分析副本需求的方法,并开发一种综合考虑需求和处理性能实现重新放置副本的算法。通过模拟实验表明,通过对集群操作中对副本的需求进行分析后,将请求量最多的数据转移到能够最快处理它们的节点上,可以将处理性能显著提高[10]。

2 算法设计(Algorithm design)

2.1 评估数据块需求

在 HDFS 管理的数据块元数据中增加 2 个元数据 C 和 T_C。在一段可配置时间(比如一个月)内,C 是数据块被查询率,即某数据块被下载的次数;T_C 是该数据块在这段时间内的平均读取时间。T_C 的计算公式参见公式(1)。其中,C 是数据块被查询次数,t_i 是数据块第 i 次被查询时的读取时间。

$$T_C = \sum_{i=1}^{C} t_i / C \tag{1}$$

DataNode 在每次读取操作后计算这些元数据,并随数据块报告传输给 Master。每当一个数据块被查询时,它相应的 C 和 T_C 都会更新。NameNode 使用一个包含所有数据块合并排序的数据表维护这两个元数据值,示例详见表 1。

表 1　数据块需求统计示例表

Tab. 1 Sample table of data chunk requirement statistics

数据块 ID	C	T_C/ms
A	150	9
B	110	5
C	40	11
D	30	6

理想状态下,统计表的每行应该是按照最大的请求数 C 对应最小的响应时间 T_C 有序排列的。然而,通过对从模拟集群中回调的数据进行分析发现,高请求率的数据块响应时间相对较长。本文的目标是将请求最多的数据块移动到能提供最优读取时间的节点中。利用元数据计算每个节点平均性能的方法详见公式(2)。其中,P_n 是节点 n 的平均性能;k 是节点 n 中数据块的数量;C_i 是节点 n 中第 i 个数据块的被查询次数;T_{C_i} 是节点 n 中查询第 i 个数据块的平均读取时间。

$$P_n = \sum_{i=1}^{k} T_{C_i} C_i / \sum_{i=1}^{k} C_i \tag{2}$$

将数据块移动到具有更快读取时间的位置，并不代表该数据块的响应时间会提高，因为平均响应时间还取决于客户端请求的性质（如发起读取请求的位置、数据处理性能等），但评估算法将继续收集被移动的数据块在其新位置上的响应时间，并重新评估是否需要将其移动到更好的位置。在数据块迭代进行几次移动后，平均查询时间会明显提高或者停滞在其最小值，这表示数据块在响应时间方面已处于最佳位置。数据块的移动操作应该在数据访问较少或没有的时候进行，这样网络带宽就不会被该操作过度占用。

2.2 算法设计

优化放置算法的流程：函数从最大查询次数开始扫描数据块，查询次数统计表，检索出请求数量最多的数据块。算法同时尝试检索具有最佳性能 P_n 的可用节点，在理想情况下，具有最高被查询数 C 的数据块将被移动到具有最优平均性能 P_n 的节点上。

优化放置算法步骤如下。第 1 步：读取数据块需求统计表 chunks_Table（*ChunkId*，T_C，C）；第 2 步：按照 C 值对统计表进行降序排序，得到表 chunks_OrdredTable；第 3 步：按顺序读取 chunks_OrdredTable 中 *ChunkId*；第 4 步：当 *ChunkId* 不为空时，循环执行第 5 步，否则，执行第 6 步；第 5 步：移动数据块至节点（*ChunkId*，GetBestNode（*ChunkId*））；第 6 步：返回 null。

上述算法中调用了获取最佳节点（GetBestNode）算法。这个算法可以为选定数据块建议一个能够提供最佳平均性能 P_n 的节点。遍历表中数据块的平均查询时间，为统计表中的每个节点计算 P_n 值，并根据 P_n 值升序排序，检查每个节点是否与待放置数据块的参数匹配。

基于以下四个标准判断节点是否匹配，这些标准均符合 Hadoop 的基本策略，即同一机架中的副本不超过两个。①该节点与待放置数据块所在节点不是同一个；②该节点有充足的可用空间；③该节点尚未存储待放置数据块的副本；④该节点与待放置数据块的其他两个副本都不在同一个机架中。

GetBestNode 算法步骤如下。第 1 步：读取待放置数据块 C_m；第 2 步：读取数据块需求统计表 chunks_Table（*ChunkId*，T_C，C）；第 3 步：计算 chunks_Table 中每个数据块的 P_n 值；第 4 步：按照 P_n 值对统计表进行升序排序得到表 chunks_OrdredTable；第 5 步：按顺序读取 chunks_OrdredTable 中 *ChunkId*；第 6 步：循环选择 chunks_OrdredTable 中下一个数据块所在节点 N；第 7 步：当节点 N 的 P_n 值小于 C_m 的 T_C 值且符合 4 个判断标准，返回节点 N；第 8 步：循环结束。

3 实验与结果（Experiments and results）

3.1 实验设置

实验中模拟了一个由 12 个节点组成的集群，结构详见图 1。使用 Hadoop 版本为 2.5.0，Hadoop 的数据块配置为 64 MB，复制因子为 3。实验中使用 12 个大小为 64 MB 的文件模拟待处理数据。该集群由 3 个机架组成，每个机架由 4 个节点组成，每个节点的存储容量等于数据块的 3 倍，即 192 MB。每个节点的性能是不同的，每个机架上 4 个节点的配置见表 2。同一机架内的带宽设为 1 GB/s，机架之间的带宽配置见图 1。

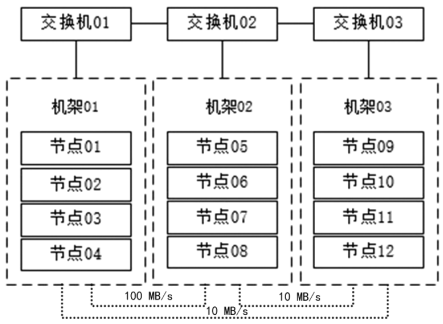

图 1 实验模拟集群结构图

Fig. 1 Cluster structure in experiment

表2 节点配置表

Tab. 2 Nodes configuration table

节点 ID	处理器	内存/GB	存储量/MB
01/05/09	1 核@2GHz	2	192
02/06/10	1 核@2GHz	4	192
03/07/11	2 核@2GHz	6	192
04/08/12	4 核@2GHz	8	192

由于每个节点最多只能存储三个数据块，所以在实验中可以迅速使节点存储达到饱和，便于更加准确地测试算法的性能。

测试作业为集群中的 12 个节点对每一个均匀分布存储在集群上的文件执行不同数量的 Map 任务。每个作业只调用一个文件，因此被请求数 C 等于执行的作业数。

实验步骤如下。第 1 步：在集群中按照 Hadoop 默认放置策略存储 12 个文件；第 2 步：运行测试作业；第 3 步：收集执行时间，并创建/更新统计表；第 4 步：计算平均性能值 P；第 5 步：如果 P 值不理想，使用算法找出数据块的建议放置点，否则，实验结束；第 6 步：移动数据块至建议放置点，并由第 2 步开始重新执行。

上述步骤会迭代几次，在每次迭代后都会计算集群的整体性能 P_n 值。通过比较默认副本放置策略和优化后的放置算法的 P 值评估优化算法的有效性。

3.2 试验结果

测试作业第一次运行后收集的结果见表 3。

表3 数据块初始统计表

Tab. 3 Initial statistics table of data chunk

数据块 Id	节点 Id	C	T_C/ms
1	1	12	20 000.33
2	2	11	21 626.64
3	3	10	23 117.40
4	4	9	20 143.00
5	5	8	16 487.00

续表

数据块 Id	节点 Id	C	T_C/ms
6	6	7	9 967.00
7	7	8	26 891.00
8	8	9	19 819.44
9	9	10	31 952.60
10	10	11	38 492.09
11	11	12	35 360.33
12	12	9	35 135.00

由于实验选用文件的大小与 Hadoop 数据块配置相同，一个文件对应一个数据块，所以每个作业只调用一个文件，C 是执行作业的次数，也就等于数据块被请求数。将表 4 的数值代入公式(2)可以计算出集群的整体性能 P_1 为 25 594.86 ms，其中的 T_C 值反映了 Hadoop 默认放置策略下的平均读取时间。

应用优化的数据放置算法后，得到的数据块放置建议见表 4。算法建议将其中 8 个数据块放置到新的位置，剩下的 4 个数据块没有建议新位置，因为将它们放置到当前集群中的任何可用节点都不会缩短任务执行时间。

表4　数据块建议放置位置表

Tab. 4 Recommended placement table of data chunks

数据块 Id	节点 Id	新节点 Id
1	1	5
11	11	5
2	2	4
10	10	4
3	3	7
9	9	7
4	4	None
8	8	None
12	12	0
5	5	None
7	7	3
6	6	None

按照算法建议位置移动数据块后，再次运行相同的测试任务，重新计算的响应时间 T_C，结果见表 5。

表5　第一次重新放置后数据块统计表

Tab. 5 Data chunks statistics table after the first replacement

数据块 Id	节点 Id	C	T_C/ms
1	6	12	20 000.33
2	5	11	21 207.73
3	8	10	23 117.40
4	4	9	20 143.00
5	5	8	16 487.00
6	6	7	9 967.00
7	4	8	29 259.00
8	8	9	19 819.44
9	8	10	22 736.60

续表

数据块 Id	节点 Id	C	T_C/ms
10	5	11	17 127.73
11	6	12	20 000.33
12	1	9	19 876.33

将表 5 的数值代入公式(2)可以计算出集群的整体性能 P_2 为 20 125.21 ms，与集群初始整体性能 P_1 相比，提高了 21.37%。在执行 5 次测试任务后，后 4 次的集群整体性能 P 值依次为 20 125.21 ms、20 098.96 ms、20 964.57 ms、20 732.94 ms。由此可看出，在第一次执行测试任务后，P 值处于稳定状态，之后的数据块位置改变几乎不会再影响集群的整体性能，那么，可以认为数据块在第一次执行优化算法后就被放置在了最佳位置。

4　结论(Conclusion)

实验数据证明，在符合 HDFS 默认数据放置算法基本规则的前提下，提出的优化的数据放置策略算法，可以使集群的整体性能提高 20%以上。并且，该算法执行一次即可使集群整体性能接近最优值，不会因为数据块频繁移动而过多占用集群的网络带宽。实验结果符合预期，优化后的数据块放置算法能够有效优化集群整体性能。

参考文献(References)

[1] 石方夏，高屹. Hadoop 大数据技术应用分析[J]. 现代电子技术，2021，44(19)：153-157.

[2] 张黎平，段淑萍，俞占仓. 基于 Hadoop 的大数据处理平台设计与实现[J]. 电子测试，2022，36(20)：74-75.

[3] 周晴红. 基于 Hadoop 的海量数据存储平台设计[J]. 无线互联科技，2022，19(17)：69-72.

[4] 王思霖. 基于 Hadoop 的日志数据处理系统[J]. 信息与电脑(理论版)，2022，34(7)：26-28.

[5] 乔永峰，孙承秀，孙玉强. 虚拟机环境下 Hadoop 集群部署与简化配置的研究与实现[J]. 工业控制计算机，2021，34(9)：130-131，133.

[6] 李玥. 基于 HDFS 的动态负载均衡方法研究[J]. 信息与电脑(理论版)，2021，33(3)：68-72.

[7] 袁爱平，陶志勇，邓河，等. 云计算环境中 HDFS 数据块存储策略研究[J]. 电脑知识与技术，2020，16(26)：33-35.

[8] 吴金坛，章超，李树楠. 大数据计算与存储分离技术实验分析[J]. 电脑知识与技术，2020，16(27)：24-27，33.

[9] 苟子安，张晓，吴东南，等. 分布式存储系统中的日志分析与负载特征提取[J]. 计算机应用，2020，40(9)：2586-2593.

[10] YORDAN K，MILKO M，TSVETELINA M，et al. Analysis and experimental study of HDFS performance[J]. TEM Journal，2021，10(2)：806-814.

作者简介：

吴　岳(1983-)，男，硕士，高级工程师. 研究领域：分布式计算，大数据处理.