计算机研究与发展
Journal of Computer Research and Development
DOI:10.7544/issn1000-1239.2016.20148327
53(2):431-442,2016

# 一种基于文件支持度的动态副本管理机制

肖中正[1] 陈宁江[1] 贾贞昊[1] 张文博[2]
[1](广西大学计算机与电子信息学院 南宁 530004)
[2](中国科学院软件研究所软件工程技术研究开发中心 北京 100190)
(village.fm@hotmail.com)

# A Dynamic Replica Management Mechanism Based on File Support Degree

Xiao Zhongzheng[1], Chen Ningjiang[1], Jia Jionghao[1], and Zhang Wenbo[2]
[1](*School of Computer and Electronic Information, Guangxi University, Nanning* 530004)
[2](*Technology Center of Software Engineering, Institute of Software, Chinese Academy of Sciences, Beijing* 100190)

**Abstract** Replication-based management schema is an important fault tolerance mechanism in large scale distributed storage systems. In response to the demand of dynamic replication management in distributed storage systems, a file popularity index named file support degree and its computation model are proposed. Within this model, file's parameters are periodically collected. By combination of self-correlation of file support degree, file hits in previous collection cycle, accessed data volume and file's grade, a model that exactly reflects files' replication requirement is built. To adapt to the variable system load, the model dynamically adjusts its parameters, making the replication decision-making to reflect real system status. Based on these work, some algorithms like load balancing, replication adjustment and replication clearing are designed. To avoid a single data storage node being overloaded, a data storage nodes' load-balance strategy is proposed. In this strategy, data storage nodes are divided into 3 groups: a holding group, an acceptable group and a begging group. There are 2 periodic procedures in the system, including replication adjusting procedure and replication clearing procedure. In replication adjusting procedure, top $P$ files are replicated to data storage nodes selected based on the load-balance strategy. Replication clearing procedure is a long-periodic procedure, because it needs many adjusting procedures to make the begging group be empty. This dynamic replication management mechanism is proven effective through the given experimentations.

**Key words** distributed storage; dynamic replication management; load balancing; file support degree; fault tolerance

**摘 要** 在大规模分布式存储系统的容错技术中,数据副本管理是一种重要机制.针对网络环境中的动态副本管理需求,建立一种文件支持度指标及其动态计算模型.该模型通过周期性数据采集,利用文件支持度的自相关性,结合文件上一采集周期访问量、访问量占比、被访问数据量以及文件级别等参数,

**收稿日期**:2014-12-08;**修回日期**:2015-03-19
**基金项目**:国家自然科学基金项目(61063012,61363003);国家科技支撑计划基金项目(2015BAH55F02);广西自然科学基金项目(2012GXNSFAA053222);广西高校优秀人才资助计划项目([2011]40);广西科学研究与技术开发计划项目(桂科软13180015,桂科攻1348020-7)
This work was supported by the National Natural Science Foundation of China (61063012, 61363003), the National Key Technology Research and Development Program of China (2015BAH55F02), the Natural Science Foundation of Guangxi (2012GXNSFAA053222), the Talents Grants Program in Colleges and Universities of Guangxi ([2011]40), and the Scientific Research and Technological Development Program of Guangxi (桂科软13180015,桂科攻1348020-7).
**通信作者**:陈宁江(chnj@gxu.edu.cn)

构建了能够较准确描述文件的动态副本需求状态模型.通过动态适应性的参数调整以适应变化的负载状态,使副本管理决策尽可能反映系统实际状态.在此基础上设计了数据结点负载均衡、副本调整、副本清理等相关算法,实现了动态副本管理的目标.通过实验验证了所设计的动态副本管理机制的有效性.

**关键词** 分布式存储;动态副本管理;负载均衡;文件支持度;容错

**中图法分类号** TP391

面向大规模分布式存储系统的容错需求,基于冗余思想的数据副本管理是常见技术.静态的副本管理通常在数据存储系统初始化时,预先设定相应的副本数量,或简单地根据文件属性给出副本创建数量的评价,但这无法应对动态环境而做出有效的负载均衡.因此,动态的数据副本管理受到更多的关注,根据系统运行时的若干参数变化触发文件副本调整,实现系统资源的有效分配.

IanFoster[1]讨论了关于高性能层次状数据网格的动态副本及缓存策略,实现并评估了3种副本管理策略——最优客户(best client)、级联复制(cascading replication)、快速传播(fast spread).网络带宽在副本创建、传输过程起到至关重要的作用[2],以上策略却没有对网络负载进行考虑.Ranganathan等人[3]建立了一种满足去中心化P2P环境下的动态模型驱动的副本机制.该模型下的任何结点拥有为存储于其上的文件创建副本的权限,并且需要事先计算在其他结点创建副本的代价和获益.这种架构较难实现全局负载均衡,只考虑副本根据参数的动态创建,而未根据参数变化对副本数量进行裁剪,可能导致存储资源浪费和负载不均衡.Anderson等人[4]提出一种数据网格下根据数据状态自动维护数据的动态副本创建和放置算法,根据访问次数以及访问时序的评价模型符合Zipf定律,但仅依赖访问次数和时间太片面,对存储系统的访问具有随机性;该算法并未提供参数修正的相关机制以及结点负载均衡的调整机制.Ding等人[5]提出了一种基于全局存储管理、全局并行任务调度框架下的副本创建和放置算法.该框架与Hadoop平台[6]以及Google云计算平台[7]相似,新数据对象的创建和放置位置是基于当前工作负载和可用存储考虑的,提出一种自我调整的数据复制算法以适应集群结点资源和数据访问模式的变化.Andronikou等人[8]提出了基于QoS感知的副本管理机制,考虑了基础设施的局部性、代价、网络带宽、文件重要程度等因素.

现有动态数据副本管理机制主要考虑的因素包括以下方面:1)动态性;2)决策计算;3)底层基础设施架构;4)副本创建代价.在动态的分布式大数据环境中,文件系统中的数据被访问的概率具有随机性和突发性,并且大部分数据访问请求的持续在线时间很短,不会长时间占用目标数据的存取权限.虽然在存储系统中存在大量文件,但是对整个系统而言并非所有的文件都会被频繁地访问,根据Carns等人[9]对文件系统的分析,90%的文件在最近30 d之内没有被访问,而55%的文件在最近64 d内未被访问.因此也需要合理地评估文件的热度.本文在前人工作基础上,面向动态的数据副本管理需求,设计文件支持度(file support degree)评价指标,并建立合理的基于时序的周期性计算模型以动态评价文件支持度.通过引入文件支持度计算模型及其参数动态调整,实现副本管理涉及的数据结点负载均衡、细粒度副本调整和副本清理等过程,提供了一种具有良好效果的优化方案.

## 1 文件支持度动态评价模型

### 1.1 文件支持度计算模型

**定义1.** 副本因子(replication factor).表示对象(文件或条块)的当前有效副本数量,记为$\gamma$.

**定义2.** 最小副本因子(minimum replication factor).是系统为文件设定的副本数量下限值,记为$\Gamma$.

当文件的副本因子减小到最小副本因子时,该文件的副本量将不会继续减少.

**定义3.** 文件支持度(file support degree).是对在第$i$个周期文件$f$在系统中的热门程度的评价,记为$\delta(i,f)$.

**定义4.** 文件级别(file grade).是用户为文件设置的一个评判其重要程度,记为$G$,且$0\leqslant G\leqslant 1$.$G$越大,文件越重要.

**定义5.** 文件访问时段.对于文件$f$而言,第$i-1$次访问与第$i$次访问之间的时间间隔为文件$f$的一个时段.

**定义 6.** 相关系数调整周期.对文件支持度相关的数据进行采集和计算的周期,记为 $\tau$.

**事实 1.** 在时段$[t_m,t_n]$之间,对于文件 $f$ 而言,$i$ 时段文件支持度 $\delta(i,f)$与 $i-1$ 时段的文件支持度 $\delta(i-1,f)$相关,并且 $\delta(i-1,f)$对 $\delta(i,f)$影响程度跟 $i-1$ 时段与 $i$ 时段之间的时差呈反相关,即过去越久的文件支持度相关数据对当前的计算影响越小.其中 $n>m$ 且 $\eta\leqslant n-m\leqslant\tau,i\in[m,n]$,$\eta$ 为最低系数样本量,$\tau$ 为相关系数调整周期.

由事实 1 可知,过去一个时段的文件支持度对当前文件支持度有很大影响.但是由于时段的划分不是等时间的,因此 $\delta(i,f)$与 $\delta(i-1,f)$直接的关联必然不是线性相关.随着时间跨度的逐渐增大,$\delta(i-1,f)$对 $\delta(i,f)$的影响显然会呈现降低的趋势[10].由事实 1 以及上述分析可得式(1):

$$\delta(i,f)\propto\frac{c}{\Delta t^{\alpha}}\delta(i-1,f),\tag{1}$$

其中,$\delta(i,f)$是文件 $f$ 于 $i$ 时段的文件支持度;$\frac{c}{\Delta t^{\alpha}}$是文件支持度依赖因子($c$ 和 $\alpha$ 是常量,用于调节该因子变化趋势),表明当前文件支持度对时间 $\Delta t$ 前的文件支持度依赖程度;$\delta(i-1,f)$是上一个时段文件 $f$ 的文件支持度.式(1)表明 $\delta(i-1,f)$与 $\delta(i,f)$的正相关性,同时又受限于支持度依赖因子.

**事实 2.** 文件 $f$ 的文件支持度与过去一个相关系数调整周期的访问量 $H$ 有关,并且跟 $H$ 在该周期系统平均访问量中的占比有关.

在 Web 应用系统中,页面资源的热门程度以及访问量是服从 Zipf 定律[11]的.这就意味着根据访问量进行的页面排名可能出现如下状况:排名第前 $k$ 位的页面量可能仅占总资源量的 $1/k$,甚至更少.同样地,在存储系统中也存在这种规律[12]:当前热门的数据集(文件支持度高)占系统总数据量较少的一部分.而描述数据集的热门程度显然不是单看其当前的访问量,还要看其排名程度.若排名越前,其访问量占比就越大.所以,文件支持度与访问量占比有极大关系.并且通过与平均访问量进行对比的简单计算方式,避免了在大量的数据集里进行排序,将会节省系统资源.根据事实 2 及以上相关叙述,可以得出式(2):

$$\delta(i,f)\propto\frac{c}{\Delta t^{\alpha}}\frac{H_i}{H_{\text{avg}}},\tag{2}$$

其中,$\delta(i,f)$是文件 $f$ 于 $i$ 周期的文件支持度;$\frac{c}{\Delta t^{\alpha}}$是文件支持度依赖因子;$H_i$ 是文件 $f$ 在 $i$ 周期的访问量;$H_{\text{avg}}$是系统 $i$ 周期的平均访问量.

式(2)表明 $\delta(i,f)$与文件访问量的排名具有正相关性,访问量排名越大对文件支持度影响越大;但同时也受到支持度依赖因子的约束,这是由于访问量的时效性问题引起的,即过去时间越久的访问量数据的参考价值越低.因此需要该依赖因子进行时效性上的限制.

**事实 3.** 文件 $f$ 的文件支持度与过去一个周期被访问的数据量的比例有关,即如果被访问的数据量越多,表明 $f$ 的文件支持度可能越高.

存储系统中的文件数据量大小波动极大.在用户访问过程中可能出现 2 种情形:1)某时刻可能存在大量用户访问某个文件极少部分数据(如仅 16 b 的标记数据),而该文件可能在 GB 级别;2)用户集中访问某文件的 1 B 数据,但该文件只有十几字节.显然,用数据访问量的大小比例来评判 $\delta(i,f)$避免了仅通过访问次数进行评判带来的片面性:大量访问可能是重复访问极少的无关紧要的数据,而这些极少的数据量访问不会对系统造成较大压力.因此数据访问量也是影响文件支持度的重要因素.根据事实 3 和上述结论,得出式(3):

$$\delta(i,f)\propto\frac{V_i}{V_f H_i},\tag{3}$$

其中,$\delta(i,f)$是文件 $f$ 于 $i$ 周期的文件支持度;$H_i$ 是文件 $f$ 在 $i$ 周期的访问量;$V_i$ 是文件 $f$ 在 $i$ 周期访问数据量,$V_f$ 是文件总数据量.式(3)表明 $\delta(i,f)$与访问数据量正相关,与 $V_f$,$H_i$ 之积负相关.

**事实 4.** 文件 $f$ 的文件支持度受文件级别的影响,文件级别越高,文件支持度的影响越大.

当文件资源持续被大量访问而产生过多副本并且系统资源吃紧时,可以通过文件级别进行调控,以避免文件资源过热而导致系统资源过度消耗的情况.根据事实 4 相关结论,得出式(4):

$$\delta(i,f)\propto G,\tag{4}$$

表明用户设置的文件级别对 $\delta(i,f)$有影响.

由式(1)~(4)可知,$\delta(i,f)$的上一周期文件支持度、文件访问量、文件访问数据量以及文件级别有正相关性.因此,为描述文件支持度的影响因素,构建向量 $\boldsymbol{z}=(\iota,\upsilon,\varphi,\zeta)$,其中各元素表示为:

$$\iota=\rho_{\delta_i,\delta_{i+1}}\frac{c}{\Delta t^{\alpha}}\delta(i-1,f);\tag{5}$$

$$\upsilon=\rho_{V,\delta}\frac{V_i}{V_f};\tag{6}$$

$$\varphi=\rho_{H,\delta}\frac{c}{\Delta t^{\alpha}}\frac{H_i}{H_{\text{avg}}};\tag{7}$$

$$\zeta=\rho_{g,\delta}G;\tag{8}$$

其中,$\rho_{\delta_{i-1},\delta_i}$是 $i$ 周期文件支持度与 $i-1$ 周期文件支持度之间的相关系数;$\rho_{V,\delta}$是 $i$ 周期数据访问量与文件支持度的相关系数;$\rho_{H,\delta}$是 $i$ 周期文件被访问次数与文件支持度的相关系数;$\rho_{g,\delta}$是文件级别与文件支持度的相关系数.

对 $z$ 各个分量进行归一化处理,采用线性函数转换进行归一化处理方法,将各分量值映射到[0,1]区间.由于各个分量(各个影响因素)对文件支持度的作用大小不尽相同,因此还需对各个分量进行加权计算,以区分影响力.

对 $\delta(i-1,f)$影响因素分量的归一化处理表示为:

$$L_\delta = \omega_\delta \times \frac{\iota - \iota_{\min}}{\iota_{\max} - \iota_{\min}}, \tag{9}$$

其中,$\iota_{\min}$是用于计算的连续样本时段集内 $\iota$ 的最小值,$\iota_{\max}$是 $\iota$ 的最大值,$\omega_\delta$ 是分量的影响力权重.

对数据访问量影响因素分量 $\upsilon$ 的归一化处理表示为:

$$L_V = \omega_V \times \frac{\upsilon - \upsilon_{\min}}{\upsilon_{\max} - \upsilon_{\min}}, \tag{10}$$

其中,$\upsilon$ 是连续样本时段集中当前的时段内数据访问量影响评价值,$\upsilon_{\min}$是最小的时段评价值,$\upsilon_{\max}$是最大的时段评价值,$\omega_V$ 是分量影响权重值.

对访问数量影响因素分量 $\varphi$ 的归一化处理表示为:

$$L_H = \omega_H \times \frac{\varphi - \varphi_{\min}}{\varphi_{\max} - \varphi_{\min}}, \tag{11}$$

其中,$\varphi_{\min}$是连续样本时段内 $\varphi$ 的最小值,$\varphi_{\max}$是 $\varphi$ 的最大值,$\omega_H$ 是分量影响权重值.

对文件级别影响因素分量 $\xi$ 的归一化处理表示为:

$$L_G = \omega_G \times \zeta, \tag{12}$$

其中,$\xi$ 是文件级别影响因子,$\omega_G$ 是分量影响权重值.

最终,得到如下的 $\delta(i,f)$计算公式:

$$\delta(i,f) = L_\delta + L_V + L_H + L_G. \tag{13}$$

### 1.2 相关系数计算及动态修正

由于带有随机性和时序性,$\delta(i,f)$与 $\delta(i-1,f)$、文件访问次数、文件数据访问量以及文件级别之间的极小可能服从高斯分布,但是根据式(1)~(4)可知,$\delta(i,f)$与各个影响因素成正相关.文件支持度的相关计算公式中涉及 3 个相关系数:$\rho_{\delta_i,\delta_{i-1}}$,$\rho_{V,\delta}$和 $\rho_{H,\delta}$,常用分析变量相关性的方法有 Pearson 法、Spearman 等.其中,Spearman 相关法不仅适用于线性相关,同时也适用于非线性相关,对样本数据是否服从正态分布无严格要求,也容许异常数据出现.文件支持度模型的相关系数符合 Spearman 相关,因此采用斯皮尔曼等级相关系数[13](Spearman rank correlative coefficient)对变量的相关性进行评价.由斯皮尔曼等级相关系数的计算公式可知,要计算 3 个文件支持度相关系数,需要对过去的时段数据进行保存用作样本.但是由于数据量巨大,不可能对所有历史数据进行保存,因此要设定限制点,对影响力小的久远数据进行丢弃,降低系统计算开销.算法 1 描述了样本采集以及相关系数修正的时序过程.

**算法 1.** SAMPLE-COL.

输入:相关系数调整周期 $T_{\rho\_\text{adj}}$,副本管理器对象 $RM$;

输出:样本集合 $SC$,相关系数 $\rho_{\delta_i,\delta_{i-1}}$,$\rho_{V,\delta}$,$\rho_{H,\delta}$.

/* 定义样本集合 */

① Def $SC$: $\{f_0:[stat_0, stat_1, \cdots, stat_n], f_1:[stat_0, stat_1, \cdots, stat_n], \cdots\}$;

② REPEAT DO

③ $req \leftarrow accept()$;

④ $f \leftarrow target(req)$;

⑤ $\Delta t \leftarrow diff(req.timestamp, f.stat_i)$;

⑥ $V \leftarrow len(req.IO)$;

⑦ $inc\_freq(f), push(f, \Delta t, V)$; /* 利用旧相关系数计算当前支持度 */

⑧ $\delta(i,f) \leftarrow L_\delta + L_V + L_H + L_G$; /* 执行副本管理操作 */

⑨ $RM.exec\_rep()$;

⑩ IF $RM$ is imbalanced

⑪ $force\_timeout(T_{\rho\_\text{adj}})$;

⑫ ENDIF

⑬ IF $T_{\rho\_\text{adj}}$ is time out

⑭ $re\text{-}compute(\rho_{\delta_i,\delta_{i-1}}, \rho_{V,\delta}, \rho_{H,\delta})$;

⑮ $adjust(SC)$; $continue()$;

⑯ ENDIF

⑰ ENDREPEAT

在算法 1 中,通过不断地对文件请求进行计算,收集请求与文件支持度相关的参数,以保持样本数据最新.但在此过程中并不是每个请求都进行文件支持度相关系数的计算,这显然会造成系统严重负载.本算法在兼顾系统压力和相关系数精确度的基础上,周期性地对相关系数进行调整;同时把系统负载作为参考依据,动态地修正相关系数.当文件经过计算并执行了副本管理之后,导致系统负载不均衡时,将会触发重新计算新的相关系数,以恰当描述文件支持度与各个影响因素之间的相关性.

## 2 动态管理过程

### 2.1 Data Node 负载均衡

当文件 $f$ 在某个数据结点上的副本数据突然收到大量访问请求时,该数据结点将会过载,可能导致结点失效等严重后果.数据结点的负载均衡显得尤为重要,由此必须对各个结点的负载状况进行收集.各个数据结点除了与元数据服务器(metadata server,MDS)保持心跳以及存储状态报告之外,还需要将自身的负载信息定期发送给副本管理器.数据结点 $DN_i$ 的负载状态可使用计算资源、主存、网络资源以及持久存储资源权衡,表示为:

$$L(DN_i) = f(\chi) + g(\psi) + h(\eta) + z(\nu), \quad (14)$$

其中,$f(\chi)$表示归一化的计算资源状态,$g(\psi)$表示归一化的主存资源状态,$h(\eta)$表示归一化的网络资源状态,$z(\nu)$则表示归一化的持久存储资源状态.

副本管理器获取到所有数据结点的负载信息之后,可以得到数据结点集群的平均负载,如式(5):

$$L_{\text{avg}}(DN) = \frac{1}{N}\sum_{i=1}^{N} L(DN_i). \quad (15)$$

根据各数据结点的负载信息和集群平均负载信息,即可构建数据结点负载分群.在实际应用中,虽然某时刻所有的数据结点中必然有负载排名较高的一些结点,但并不能表明这些结点一定处于重载状态下,因为可能当前所有结点的请求量都很少.同样地,负载排名较后的结点也并不能表明结点一定轻载,比如当系统极度繁忙时所有结点都接受到大量请求.即便是普通情况下,连续的数据结点可能负载均衡差别极小却被分配到不同的结点群里.因此,不能单一以负载排名顺序对各个数据结点进行划分.根据数据结点负载信息,将结点划分为 3 个结点群:保持群(holding)、接收群(acceptable)和请求群(begging),分别记为 $DG_{\text{H}}$,$DG_{\text{A}}$,$DG_{\text{B}}$.$DG_{\text{H}}$ 里的结点不允许发出任何迁移请求,同时也不接收任何迁移请求;$DG_{\text{A}}$ 里的结点可以接收迁移请求,但不允许发出迁移请求;$DG_{\text{B}}$ 里的结点允许发出迁移请求,但不允许接收迁移请求.对 3 个结点群有如下约束:

**约束 1.** $DG_{\text{A}}$ 和 $DG_{\text{B}}$ 可以为空,但 $DG_{\text{H}}$ 任何情况下不允许为空,并且 $DG_{\text{A}} \cap DG_{\text{B}} = \varnothing$,$DG_{\text{A}} \cap DG_{\text{H}} = \varnothing$,$DG_{\text{H}} \cap DG_{\text{B}} = \varnothing$.

约束 1 可以有效避免极端情况下大量重载结点向重载结点迁移、轻载结点向轻载结点迁移的状况.

**算法 2.** DN-GROUPING.

输入:数据结点集 $DS$,负载变化率限制 $\kappa$;

输出:保持群 $DG_{\text{H}}$,接收群 $DG_{\text{A}}$,请求群 $DG_{\text{B}}$.

① Def $DG_{\text{H}}$,$DG_{\text{A}}$,$DG_{\text{B}}$;

② Sort $DS$ with loads;

③ $L_{\text{avg}} \leftarrow \frac{1}{N}\sum_{i=1}^{N} L(DN_i)$;

④ $DN_a \leftarrow (L(DN_i) \xrightarrow{\text{neareset}} L_{\text{avg}})$; /* 获得负载最接近 $L_{\text{avg}}$ 的结点 */

⑤ FOR $i=a$ to $n$ DO

⑥ $K_i \leftarrow slope(DN_a, \cdots, DN_i)$; /* 计算 $DN_a$ 向高负载扩散的结点 */

⑦ IF $|K_i| < \kappa$

⑧ $push(DN_i, DG_{\text{H}})$;

⑨ ELSE

⑩ $push(DN_i, \cdots, DN_n, DG_{\text{B}})$,BREAK; /* 满足负载变化限制的 $DG_{\text{B}}$ */

⑪ ENDIF

⑫ ENDFOR

⑬ FOR $i=a$ to 0 DO /* 对轻载方向的结点做同样处理 */

⑭ $K_i \leftarrow slope(DN_a, \cdots, DN_i)$;

⑮ IF $|K_i| < \kappa$

⑯ $push(DN_i, DG_{\text{H}})$;

⑰ ELSE

⑱ $push(DN_i, \cdots, DN_0, DG_{\text{A}})$,BREAK;

⑲ ENDIF

⑳ ENDFOR

算法 2 描述了数据结点根据负载状况进行分群的过程.它并未采用简单的根据负载量进行绝对的划分,而是根据集群的平均负载状况,从中间结点进行逐个分离.首先要对结点按负载进行排序,找到最接近平均负载状况的结点 $DN_a$,将结点加入 $DG_{\text{H}}$.这种做法具有合理性:对于普通情况,$DN_a$ 负载适中,自然属于保持群;当出现所有结点均为重载的时候,各结点都无法接收迁移,而又没有接收结点,因此 $DN_a$ 纳入保持群;当所有结点均轻载时,各结点不需要请求迁移,同时也没有请求结点,$DN_a$ 同样也纳入保持群.确定了 $DN_a$ 后,向高负载的逐个筛选,考察其相对于前面结点负载变化率.如果负载变化率在可接受范围,则将其纳入 $DG_{\text{H}}$,否则将后续结点全部纳入 $DG_{\text{B}}$.同样地,向低负载结点逐个考察,直到得到 $DG_{\text{A}}$.

通过上述分析,明确了数据结点的负载划分.

图1展示了数据结点上的周期性负载均衡调度过程.在数据结点上有3个周期性计算工作:周期性向MDS发送心跳信息和状态报告($T_{\text{heartbeat}}$)、周期性向副本管理器发送负载信息($T_{\text{rm\_report}}$)、周期性接收副本管理器的调度信息($T_{\text{LB}}$).在$T_{\text{LB}}$周期内,副本管理器获得了所有数据结点的负载信息,并依据负载信息对数据结点进行了分群.当前结点获取到分群信息,若发现本结点属于$DG_{\text{B}}$,则表明结点的负载过重,需要进行副本的迁移工作.迁移的主要步骤是:将文件支持度过高的文件选出,并在$DG_{\text{A}}$中创建其副本,然后释放本地资源.在选择创建副本的结点过程中,可能从$DG_{\text{A}}$选取多个结点,而不是单个.这样避免了迁移后导致$DG_{\text{A}}$中的结点负载突然加重的情况.

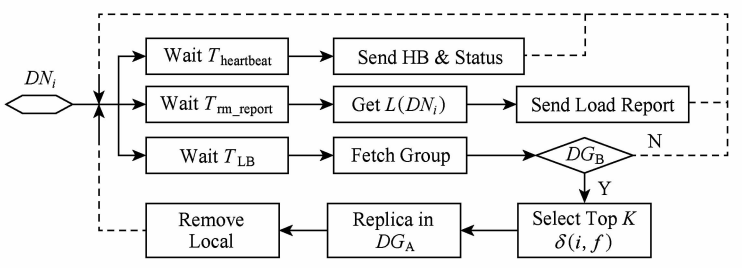



Fig. 1 Periodic computation of data node.

图1 数据结点上的周期性计算

### 2.2 副本调整

虽然在$T_{\text{LB}}$周期内可以对$DG_{\text{B}}$上的数据结点的文件支持度较大文件进行副本调整,但并未从每个文件的角度去考虑.有可能在$DG_{\text{A}}$和$DG_{\text{H}}$都存在大量文件支持度较高、但副本因子较小的文件,还需要建立以文件支持度为驱动的数据副本调整策略,更全面保障数据结点的负载均衡.

对于文件$f$在相关系数调整周期$T$下,其支持度为$\delta(T,f)$.为了根据文件支持度和副本因子来调节$f$的副本数量,定义如下调节公式:

$$\gamma^* = \Gamma \times \frac{\delta(T,f)}{\frac{1}{W}\sum_{i=0}^{W}\delta(T,f_i)}, \tag{16}$$

其中,$\gamma^*$是文件$f$预期副本因子,$\Gamma$是最小副本因子,$W$是单位支持度计算窗口.

式(16)使用支持度最小的$W$个文件的支持度均值作为单位支持度,这样避免仅使用单个支持度(最小支持度文件)作为单位支持度的片面影响.

得到与支持度相关的预期副本因子后,即可对文件的副本进行相应调整.算法3描述了以文件支持度为驱动的副本调整策略,它周期性地执行基于文件支持度驱动的副本调整,这个周期通常是离线的,并且是长周期,因为这种调整会消耗较多系统资源.

**算法3.** ADJ-SP-DRIVEN.

输入:文件集$SF$,计算周期$T_{\text{adj}}$,单位支持度计算窗口$W$,调整文件比例$P$,最小副本因子$\Gamma$;

输出:总共调整的文件数量$n$.

① Def $n=0$;

② REPEAT DO

③ $wait(T_{\text{adj}})$;

④ $\delta_{\text{unit}} \leftarrow \frac{1}{W}\sum_{i=0}^{W}\delta(T,f_i)$; /* 计算单位支持度 */

⑤ FOR $f$ IN top $P$ $SF$ DO /* 对较高支持度的文件进行调整 */

⑥ $\gamma^* \leftarrow \delta(T,f)/\delta_{\text{unit}}$;

⑦ $\Delta\gamma \leftarrow \Gamma \times \gamma^*$;

⑧ IF $\Delta\gamma > 0$

⑨ create $\Delta\gamma$ replications for $f$; $n$++; /* 为文件创建额外副本 */

⑩ ENDIF

⑪ ENDFOR

⑫ ENDREPEAT

另外,算法3并不是对所有的文件进行副本调整,而是选取支持度较高的文件进行处理,因为这些文件在当前看来价值更大.由前文分析结论知道,存储系统的访问规律是满足Zipf定律的,因此支持度很高的文件只占系统的少量,在一定程度上可降低调整过程中的不必要计算.

### 2.3 副本清理

在连续多个$T_{\text{LB}}$周期内,可能出现$DG_{\text{B}}$为空的情况,但此时可能存在许多文件的副本因子很大,但是其文件支持度却一直处于较低状态.这种情况在系统运行一定时间之后会非常多,导致存储资源的浪费,并给系统带来更大的维护开销.因此需启动周期性清理机制,如算法4所示.

**算法 4.** LB-REP-CLEAR.

输入:文件集 $SF$,清理周期 $T_{clear}$,惰性因子 $FC_{idle}$,清理比例 $P$;

输出:清理的总副本 $n$.

① Def $n=0$;

② REPEAT DO

③ $wait(T_{clear})$;

④ $\delta_{unit} \leftarrow \sum_{i=0}^{P\times Len(SF)} \delta(T,f_i)$; /* 计算临时单位支持度 */

⑤ FOR $f$ IN bottom $P$ $SF$ DO /* 选取支持度较小的文件 */

⑥ $I \leftarrow \delta(T,f)/(atan(\gamma_f)\times 2/\mathrm{PI})$; /* 计算空闲因子 */

⑦ IF $I > FC_{idle}$

⑧ $set(\gamma_f,\delta_{unit})$;

⑨ $remove(\gamma_f - \delta_{unit})$ replications of $f$; /* 删除多余的副本 */

⑩ ENDIF

⑪ ENDFOR

⑫ ENDREPEAT

算法 4 同样通过周期性的扫描,仅对低支持度的文件进行处理.目标是对空闲因子(文件支持度与归一化副本因子的比值)较大的文件进行副本清理.而当发现文件需要清理时,将其副本因子设置为当前清理比例下所有文件支持度的均值.由于选择清理的这些文件均为低支持度文件,选择均值作为新副本因子是合理的.

## 3 测 试

本文基于 HDFS[14] 的实验平台对本文方法进行原型实验,实验环境的配置部署如表 1 所示.我们主要测试新的动态副本管理机制对存储系统的负载均衡、存储资源利用率、响应时间等影响,以及分析动态副本创建的代价.

**Table 1 Experimental Environment Configuration**

**表 1 实验环境配置**

| Component | Number | Hardware | Software Settings | Other Configuration |
|---|---|---|---|---|
| MDS | 7 | Inspur NF8560M2×2 | CentOS | $vNodes=150$ $Hash=[0,2^{36})$ |
| Data Node | 18 | Sugon I450-G10×9 | OpenJDK Python 2.7 | $T_{heartbeat}=3$ s |
| Replica Manager | 2 | Sugon I450-G10×1 | NFS v4.0 | |

如图 2 所示,实验设计了 4 种不同的模拟集群访问场景:1)请求量在测试周期保持高压状态(Trace1),模拟系统在持续高负载状态下的访问过程;2)请求量持续增加(Trace2),模拟系统从空闲状态开始,用户请求量逐渐增加,工作负载逐渐增加,进入到高负载状态;3)请求量持续减少(Trace3),模拟是用户请求量逐渐减少,系统进入到相对空闲状态;4)请求量跳跃性突变(Trace4),模拟短时间内大批量的任务同时完成或者同时启动,导致系统负载抖动厉害.每种 Trace 运行时间远大于数据结点调整周期 $T_{LB}$、副本调整周期 $T_{adj}$ 和副本清理周期 $T_{clear}$.

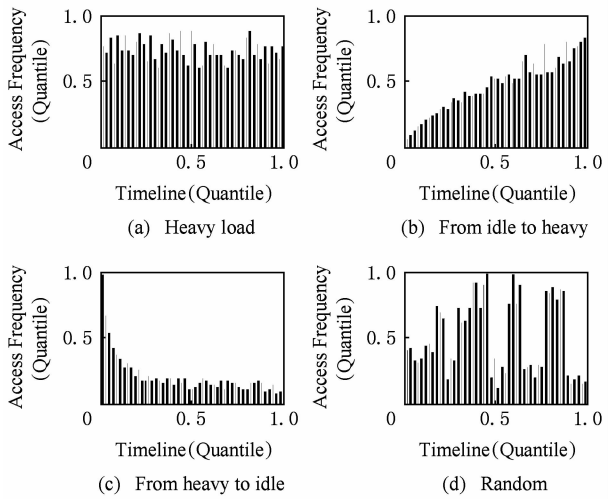



Fig. 2 Four kinds of system requests traces.

图 2 4 种系统请求场景

实验在以上配置基础上作了如下假设:

1)假设实验测试期间 MDS 结点都是可靠的,不会发生失效事件,也不会有新的结点加入 MDS 集群.同样,数据结点集群结构也保持不变,所有网络连接安全可靠,不会发生数据丢失.

2)默认副本数量为 3,将 2 个副本放置同一机架,而第 3 个副本放置于不同的远端机架,同机架副本可以保证较快数据读取,而远端机架副本则起到失效恢复作用.因此实验中的静态副本机制的副本数量也设置为 3.而对于文件支持度模型而言,初始时候假设的访问量都很低,因此文件读写量很少,从而设置 2 个副本保证到失效恢复即可,此时无需考虑读写性能.随着模型逐渐进入稳定状态,其副本量会自动调整,以适应不同支持度的可靠性和读写性能.

3)数据结点集群负载均衡度计算方法为

$$LBD=\sqrt{\frac{1}{N}\sum_{i=0}^{N}(L(DN_i)-L_{avg}(DN))^2}, \quad (17)$$

其中,$L(DN_i)$为第 $i$ 个数据结点的工作负载量,$L_{avg}(DN)$为数据结点平均工作负载量.

4) 读密集型测试过程包含主要是读操作,随机加入少量写操作(约 5%).写操作一般需要先对元数据进行查询操作,因此假设写密集型测试过程包含约 30%的读操作.

### 3.1 副本管理测试

在仿真实验过程中,通过考察文件支持度驱动的模型平均副本数量的变化,对动态副本调整的效果进行分析和评判.数据结点集群的负载均衡直接被动态副本调整影响.当文件支持度高的文件通过副本调整机制创建额外副本后,可以降低目标数据结点工作负载,从而平衡集群工作负载.

1) 副本数量动态调整测试

图 3 显示了在 4 种实验场景下平均副本数量的动态调整过程.由于 Trace1 整个过程都是处于高请求量的状态(图 2(a)),因此在系统初始化后马上收到大量的数据访问请求,导致很多文件的支持度急剧上升.但根据算法 1 所描述的相关系数计算过程可知,系统在初始化时的样本数据为空,虽然文件访问量极大,但计算得到支持度无法反映实际情况.所以在 Trace1 开始的短时间内都呈现平均副本因子逐渐上升,而非跳跃上升的情况.

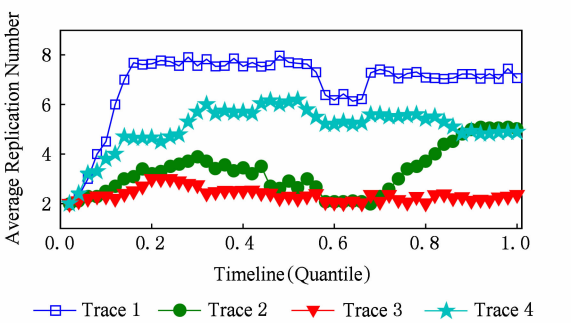



Fig. 3 Effects of dynamic replication adjustment.

图 3 动态副本调整效果

其他 Trace 情况也一样,只是由于访问量变化不同导致变化幅度不同.Trace1 进入稳定阶段后,平均副本因子保持在很小范围波动.当时间线在 0.6 左右时,出现了迅速减少的现象.这是由于在该时刻启动了周期为 $T_{clear}$ 的副本清理过程,导致平均副本因子出现突然的减少.过了该周期后,由于密集的访问量到来,使得平均副本因子又开始增加,但再次到达平稳阶段的大小比副本清理前要小.Trace2 过了初始化后一直处于缓慢增加状态,然后由于前段的访问量很少,平均副本因子波动略微减少.遇到副本清理周期迅速减少,但此后由于 Trace2 访问量增到非常大,逐渐进入 Trace1 初始化后的过程.Trace3 虽然开始就收到非常大的访问量冲击,但逐渐减少,根据式(2)可知,大访问量对支持度影响越来越弱.所以 Trace3 初始化阶段只有缓慢的增加,并且此后在 $\Gamma$ 线附近浮动.Trace4 显示的是访问量突变的情况,包含 4 个突变低谷,但是图 3 显示的 Trace4 平均副本因子动态调整过程并没有因为访问量的突变而呈现跳跃性变化,这是由于支持度计算模型依赖过去一定时段的数据,并且综合 $L_\delta$, $L_V$, $L_H$, $L_G$ 这 4 种因素进行评价.

2) 数据结点负载均衡测试

仿真实验以 Trace1 作为测试过程,对文件支持度模型驱动的动态调整策略(SPD)、静态副本机制(static)和无副本机制进行数据结点负载均衡效果的比较,如图 4 所示.无副本机制初始化后 $LBD$ 较快增加,并且出现大幅波动;整个过程没有一个相对平稳的阶段,并且 $LBD$ 相比其他 2 个大很多,负载均衡状况受访问变化的影响巨大.而静态副本机制相比无副本状态 $LBD$ 小很多,显然是因为有多个副本可以选择使得热点数据请求被系统负载均衡模块均分了.静态副本机制有相对平稳的时段,但也出现了较多突变的状况.SPD 在初始化阶段由于计算得到的相关系数失实,$LBD$ 也快速增大,甚至超过静态策略.此后 $T_{adj}$ 和 $T_{LB}$ 调整周期的作用,由于 $LBD$ 进入相对稳定的过程,并且 $LBD$ 较小,系统没有发生巨大的负载失衡状况.

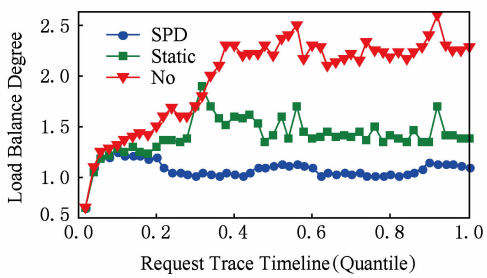



Fig. 4 Dynamic load balancing of data nodes(Trace1).

图 4 数据结点动态负载均衡(Trace1)

3) 存储利用率测试

图 5 显示了实验过程记录的不同 Trace 下各个副本机制存储利用率的比较.显然无副本机制在 4 个 Trace 下的平均存储利用率都较低,Trace1 最大达到 18%左右,Trace3 只有 10%不到.静态副本机制则比无副本机制存储利用率高很多达到 40%~50%,而具有动态调节能力的 SPD 比静态策略高.

对比图 3 可以看出,Trace1 和 Trace2 的平均副本因子比 Trace3,Trace4 高出不少;Trace2 和 Trace3 更接近静态副本机制.所以 Trace1 和 Trace4 的存储利用率 SPD 比静态策略高约 15%,而 Trace2 和 Trace3 则仅高出 5%左右.

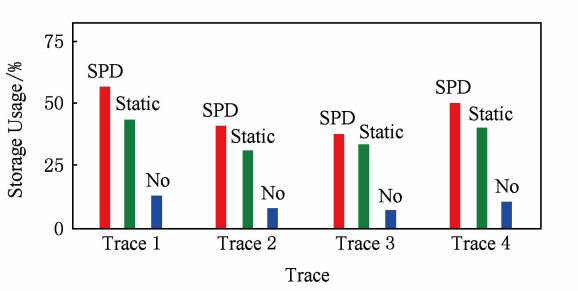



Fig. 5 Comparison of storage usage.

图 5 存储利用率比较

### 3.2 响应时间测试

系统响应时间的仿真设计,是通过对 Trace1~Trace4 测试过程记录系统负载量与响应时间,然后将系统负载归一化处理,并计算 4 个 Trace 响应时间的均值.测试过程分为读密集型测试和写密集型测试,读密集型测试 90%以上的请求量是读请求,而写密集测试则是 90%以上的测试为写请求.如图 6 所示,在读密集型的测试 Trace 中,随着系统工作负载逐步增加,静态策略和无副本机制系统响应时间缓慢增加;而 SPD 策略在测试初始化后系统工作负载不大的情况下,系统响应时间就突然增大,超过了另外二者.这是由于此时 SPD 启动了副本动态创建的机制,消耗了一定的系统资源,并且此时多数文件仅有 2 个副本,从而导致响应时间突然激增.当初始副本创建完毕,$T_{adj}$,$T_{LB}$,$T_{clear}$等周期计算完全运行起来之后,系统响应时间逐渐下降,并且此后伴随着小幅波动一直处于平稳状态.而无副本机制在系统负载量达到临界值之后,系统响应时间激增,直到系统响应超时.而静态策略同样存在这个临界值,只是由于具有 3 个副本,其临界值出现较晚.显然,在

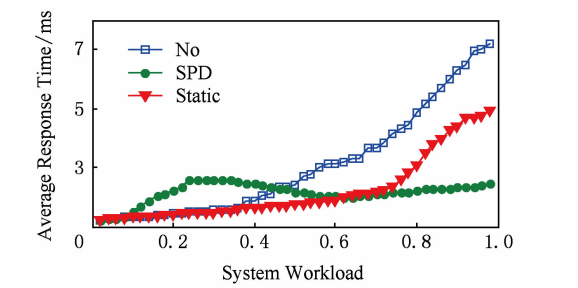



Fig. 6 Response time of read-intensive systems.

图 6 读密集型系统响应时间

系统负载不断加大情况下,对于读密集型 Trace 静态方法没有提供自适应机制,是无法满足性能提升需求的.

图 7 反映写密集型的 Trace 测试结果,显然比读密集型的 Trace 响应时间大很多.同样刚初始化后,对于 SPD 策略,由于启动副本相关机制消耗一定系统资源而导致响应时间突然增大.此后随着系统工作负载增加,SPD 策略响应时间并非如同写密集型 Trace 那样减少,而是继续保持增加,当增加到临界值(本实验条件下约为 0.66)后开始激增,系统响应时间越来越大.对于静态策略,在初始化后持续增加负载期间都保持较低的响应时间,直到达到临界值;同样无副本机制也是在临界值前缓慢增加,只不过其临界值大于写密集型时的临界值,较早地就进入了响应时间激增阶段.

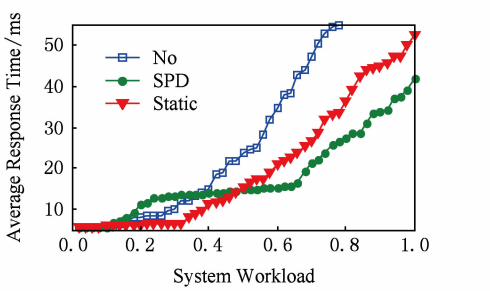



Fig. 7 Response time of write-intensive systems.

图 7 写密集型系统响应时间

从上面分析可以看出,对于读密集型应用场景,SPD 策略具有良好的系统响应能力;对于写密集型的应用场景,在系统工作负载达到临界值之前,SPD 策略具有良好的系统性能提升,而系统负载超过该值后,SPD 策略会失效.这是因为系统工作负载达到极限情况下,连续大量的写操作会导致带宽、I/O 以及存储空间等资源极度紧张,而可能有的结点没有足够的资源进行副本调整,从而导致无法完成预期的动态调整.

### 3.3 模型动态调整测试

由 3.2 节和算法 1 的分析得出,相关系数调整周期 $T_{\rho_adj}$对系统响应时间和数据结点的负载均衡具有很大的影响.如果 $T_{\rho_adj}$过小,导致短时间内系统需要消耗大量的运算资源去计算相关系数以及调整文件副本因子;而如果 $T_{\rho_adj}$过大,则会导致很长时间内相关系数无法更新,这样就无法反映出当前的系统状态,从而导致负载失衡、系统性能下降等一系列问题.

图 8 显示了在实验环境下，$T_{\rho_adj}$在不同取值下对系统响应时间和数据结点集群负载均衡的影响. 在 $T_{\rho_adj}$很小时，最能反映最新的负载状况，并且得到最佳的副本调整结果(因此 $LBD$ 很小)；随着 $T_{\rho_adj}$增大，$LBD$ 逐渐增大，因为当前的相关系数预测的系统状态与实际情况差距越来越大. 对于系统响应时间，在 $T_{\rho_adj}$很小时，由于频繁地、过多地消耗系统资源，导致其值非常大；但随着 $T_{\rho_adj}$进一步增大时，响应时间迅速降低，并达到最佳响应时间；$T_{\rho_adj}$继续增大会出现相关系统计算出现偏差越大，系统负载均衡效果不理想，从而使得响应时间逐步增大. 在图 8 中，$a$ 是 $LBD$ 和系统响应时间的第 1 个交点，$b$ 是响应时间最低点，$T_{\rho_adj}$取值$[a,b]$之间的任意值可以得到负载均衡和响应时间的较好平衡点.

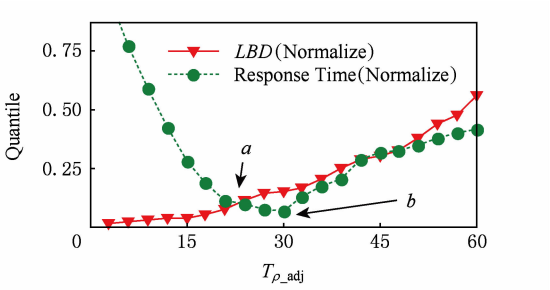



Fig. 8 Effects of adjustment cycle.

图 8 调整周期变化的影响

## 4 讨论和分析

表 2 从架构、可扩展性等方面将本文方法(SPD)与其他副本管理相关工作进行了定性比较. 典型的副本管理架构有 P2P、多层次架构、兄弟树、图以及一般的数据网格结构. 副本创建的决策主要有集中决策和分布式决策 2 种方式，集中式决策会消耗较多系统资源，可能导致系统瓶颈；而分布式决策将决策计算分散到多个结点，但可能导致重复的复制.

**Table 2 Comparison of Replication Management Mechanisms**

**表 2 副本管理机制定性比较**

| Items | Decision Making | Architecture | Cost Awareness | Scalability | Features | Scenarios |
|---|---|---|---|---|---|---|
| Abdullah et al.[15] | Decentralized | P2P Grid | NO | YES | *N*-hop Neighbour Node Placement | P2P-decentralized Data Grid |
| Yuan et al.[16] | Decentralized | Muti-tier Grid | NO | NO | Local Optimization | Multi-tier Data Grid |
| Rasool et al.[17] | Centralized | Sibling Tree | NO | NO | Replica Re-locate, Two-way Replication | Multi-tier Sibling Tree Architecture |
| Ding et al.[5] | Centralized | General Grid | YES | NO | Self Tuning | Storage Limited |
| Lei et al.[18] | Decentralized | Graph | NO | NO | Low Data Missing | Single-tier |
| Perez et al.[19] | Centralized | Multi-tier | YES | YES | Parallel I/O | Multi-tier Storage Limited |
| Sashi et al.[20] | Centralized | Multi-tier | YES | NO | Modified BHR | Multi-tier Storage Limited |
| Bsoul et al.[21] | Centralized | Graph | YES | NO | Enhanced Fast Spread | General Graph |
| SPD | Centralized | Master/Slave | YES | YES | Self-adaptive | Read-intensive, Tightly Coupled Arch |

大量的动态副本机制基于阈值决策，通常选择的决策因素有文件访问历史记录、存储余量、I/O 量等. Yuan[16]的工作以各结点存储余量和带宽可用量作为评判因子，实现局部访问优化. PHFS[22]扩展了快速传播策略，通过收集文件访问记录，然后使用分类和聚类等数据挖掘工具找出文件聚集信息，预测文件未来文件关联程度，以进行副本调整. 该方法需要的计算量巨大，实现复杂，故对于实时性要求高的系统不适用. 基于文件访问热度的方法有 Shorfuzzaman[23]，Chang 等人[24]的工作，它们在基于文件访问量越大、未来访问概率越大的假设前提下，定期收集访问信息并做分析计算，然后清除这些数据开始新的周期. 仅依靠文件访问历史数据的挖掘来预测未来文件访问概率的方法，对于访问过程的模型稳定的系统非常适用；但若系统访问的随机性大，则它们会出现预测不准确、系统资源消耗大等问题.

本文提出的 SPD 模型针对这些缺陷进行改进，

将访问量、访问数据量、文件基本以及前一周期指标作为当前评价指标参考因素,周期性地对模型参数进行调整,以适应当前时段内系统访问的分布状态. SPD 模型计算资源开销主要通过 $T_{\mathrm{LB}}$, $T_{\mathrm{adj}}$, $T_{\rho\_\mathrm{adj}}$, $T_{\mathrm{clear}}$ 等 4 个参数集中体现. $T_{\mathrm{LB}}$, $T_{\mathrm{adj}}$ 和 $T_{\mathrm{clear}}$ 通常设置为较大周期,对系统实时开销要求较小;$T_{\rho\_\mathrm{adj}}$ 是影响开销最敏感的参数. 由于 $T_{\rho\_\mathrm{adj}}$ 值设置极小时会使得系统开销巨大,但是设置过大时对系统的优化效果则极差. 在实际应用系统中,测试出 $T_{\rho\_\mathrm{adj}}$ 的平衡点区间后,将使系统副本调整效果达到最佳效果.

基于网格或 P2P 的架构通常适合于模型稳定的存储系统,因为这类系统的架构对于随机性较大的系统预测准确性低. 本文所提出的动态副本管理机制,在基于决策计算、空间利用率、系统响应时间和负载均衡等因素情况下,通过自适应调整实现动态副本调整策略. 实验表明该模型的负载均衡能力较好,能够在多种应用场景下保持稳定副本调节能力并具有较好效果. 对于读密集型的应用场景,该模型系统响应能力较好;但对于写密集型应用,极端情况下会导致失效.

## 5 结束语

本文在基于文件热度的动态副本机制相关研究成果上,提出基于文件支持度的自适应副本管理机制. 主要贡献在于:1)建立了文件支持度的计算模型,在此基础上,从数据结点层次提出了数据结点分群方法,以有助于提供副本数据结点的负载均衡能力;2)从文件的角度提出以文件支持度为驱动的副本调整、副本清理等副本管理算法,通过周期性对文件副本因子等因素进行优化,以达到更佳的效果. 通过实验验证了本文提出的方法在读密集型的应用场景下性能良好. 未来的工作将针对写密集型的应用表现欠佳的问题,将读/写操作量加入模型,研究根据读/写场景调整副本决策改进机制.

## 参 考 文 献


[1] IanFoster K R. Design and evaluation of dynamic replication strategies for a high-performance data grid [C] //Proc of Int Conf on Computing in High Energy and Nuclear Physics. Philadelphia, PA: IOP Publishing, 2001: 1-16

[2] Sashi K, Thanamani A S. A new replica creation and placement algorithm for data grid environment [C] //Proc of the 2010 Int Conf on Data Storage and Data Engineering. Los Alamitos, CA: IEEE Computer Society, 2010: 265-269

[3] Ranganathan K, Iamnitchi A, Foster I. Improving data availability through dynamic model-driven replication in large peer-to-peer communities [C] //Proc of the 2nd IEEE/ACM Int Symp on Cluster Computing and the Grid. Los Alamitos, CA: IEEE Computer Society, 2002: 376-376

[4] Anderson E. Capture, Conversion, and Analysis of an Intense NFS Workload [C] //Proc of the 7th USENIX Conf on File and Storage Technologies. Berkeley, CA: USENIX, 2009: 139-152

[5] Ding Ying, Lu Ying. Automatic data placement and replication in grids [C] //Proc of the Int Conf on High Performance Computing. Los Alamitos, CA: IEEE Computer Society, 2009: 30-39

[6] Taylor R C. An overview of the Hadoop/MapReduce/HBase framework and its current applications in bioinformatics [J]. BMC Bioinformatics, 2010, 11(S1): 1-6

[7] Ghemawat S, Gobioff H, Leung S T. The Google file system [C] //Proc of the 19th ACM Symp on Operating Systems Principles. New York: ACM, 2003: 29-43

[8] Andronikou V, Mamouras K, Tserpes K, et al. Dynamic QoS-aware data replication in grid environments based on data "importance" [J]. Future Generation Computer Systems, 2012, 28(3): 544-553

[9] Carns P, Harms K, Allcock W, et al. Understanding and improving computational science storage access through continuous characterization [J]. ACM Trans on Storage, 2011, 7(3): 1-25

[10] Hua Y, Jiang H, Zhu Y, et al. SmartStore: A new metadata organization paradigm with semantic-awareness for next-generation file systems [C] //Proc of the Conf on High Performance Computing Networking, Storage and Analysis. New York: ACM, 2009: 1-12

[11] Almeida V, Bestavros A, Crovella M, et al. Characterizing reference locality in the WWW [C] //Proc of the 4th Int Conf on Parallel and Distributed Information Systems. Los Alamitos, CA: IEEE Computer Society, 1996: 92-103

[12] Rabinovich M, Rabinovich I, Rajaraman R, et al. A dynamic object replication and migration protocol for an Internet hosting service [C] //Proc of the 19th IEEE Int Conf on Distributed Computing Systems. Los Alamitos, CA: IEEE Computer Society, 1999: 101-113

[13] Gong Shuming, Zhu Hailing. Applied Statistics [M]. 3rd ed. Beijing: China WaterPower Press, 2010 (in Chinese)
(龚曙明, 朱海玲. 应用统计学[M]. 3 版. 北京: 中国水利水电出版社, 2010)

[14] Kuang H, Radia S, Shvachko K. The Hadoop distributed file system [C] //Proc of 2010 IEEE/NASA Goddard Conf on Mass Storage Systems and Technologies. Los Alamitos, CA: IEEE Computer Society, 2010: 1-10

[15] Abdullah A, Othman M, Ibrahim H, et al. Decentralized replication strategies for P2P based scientific data grid [C] //Proc of the 2008 Int Symp on Information Technology. Los Alamitos, CA: IEEE Computer Society, 2008: 1-8

[16] Yuan Yulai, Wu Yongwei, Yang Guangwen, et al. Dynamic data replication based on local optimization principle in data grid [C] //Proc of the 6th Int Conf on Grid and Cooperative Computing. Los Alamitos, CA: IEEE Computer Society, 2007: 815-822

[17] Rasool Q, Li Jianzhong, Zhang Shuo. Replica placement in multi-tier data grid [C] //Proc of the 8th IEEE Int Conf on Dependable, Autonomic and Secure Computing. Los Alamitos, CA: IEEE Computer Society, 2009: 103-108

[18] Lei Ming, Vrbsky S V, Hong Xiaoyan. An on-line replication strategy to increase availability in data grids [J]. Future Generation Computer Systems, 2008, 24(2): 85-98

[19] Pérez J M, García-Carballeira F, Carretero J, et al. Branch replication scheme: A new model for data replication in large scale data grids [J]. Future Generation Computer Systems, 2010, 26(1): 12-20

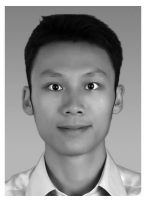
[20] Sashi K, Thanamani A S. Dynamic replication in a data grid using a modified BHR region based algorithm [J]. Future Generation Computer Systems, 2011, 27(2): 202-210

[21] Bsoul M, Al-Khasawneh A, Abdallah E E, et al. Enhanced fast spread replication strategy for data grid [J]. Journal of Network and Computer Applications, 2011, 34(2): 575-580

[22] Khanli L M, Isazadeh A, Shishavan T N. PHFS: A dynamic replication method, to decrease access latency in the multi-tier data grid [J]. Future Generation Computer Systems, 2011, 27(3): 233-244

[23] Shorfuzzaman M, Graham P, Eskicioglu R. Popularity-driven dynamic replica placement in hierarchical data grids [C] //Proc of the 9th Int Conf on Parallel and Distributed Computing, Applications and Technologies. Los Alamitos, CA: IEEE Computer Society, 2008: 524-531

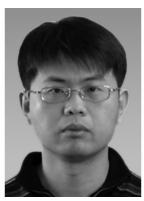
[24] Chang R S, Chang H P. A dynamic data replication strategy using access-weights in data grids [J]. The Journal of Supercomputing, 2008, 45(3): 277-295



**Xiao Zhongzheng**, born in 1988. Received his bachelor degree in Guangxi University. His main research interests include distributed computing and software engineering.

**Chen Ningjiang**, born in 1975. Professor and PhD. His main research interests include distributed computing and software engineering.

**Jia Jionghao**, born in 1989. Received his bachelor degree in Taiyuan University of Technology. His main research interests include distributed computing and software engineering.



**Zhang Wenbo**, born in 1976. PhD. Professor and senior engineer. His main research interests include distributed computing and software engineering