云存储系统 Master 节点故障动态切换算法

马玮骏,王 强,何晓晖,冯 径,马 强
(解放军理工大学,江苏 南京 211101)

摘 要:为了解决大规模云存储系统中 Master 节点发生故障导致存储服务不可用的问题,建立了面向云存储系统管理节点发生故障时的故障影响分析模型。该模型以存储服务可用性、数据可靠性和数据可用性为分析目标,通过故障状态、管理节点实时状态以及管理节点故障的限制条件三个维度对故障影响进行分析,为恢复故障提供了有效的方法依据。同时,基于故障影响分析模型,提出了一种基于消息的 Master 节点故障动态切换算法—DSA-M。该算法通过基于序号的优先级策略实现了 Master 节点动态申请和切换,保证了云存储服务的高可用性。测试结果表明,DSA-M 算法能够在 Master 节点发生故障时自动进行 Master 节点的切换和接管,恢复云存储服务的运行;通过控制故障检测周期,能够使得 DSA-M 算法的性能保持在相对稳定的区间内,随失效时刻的适应性也比较强。

关键词:云存储系统;Master 节点;故障检测;元数据;动态切换
中图分类号:TP301.6 **文献标识码**:A **文章编号**:1673-629X(2017)09-0085-07
doi:10.3969/j.issn.1673-629X.2017.09.019

Dynamic Switching Algorithm for Master Node in Huge Cloud Storage System

MA Wei-jun, WANG Qiang, HE Xiao-hui, FENG Jing, MA Qiang
(PLA University of Science and Technology, Nanjing 211101, China)

Abstract: In order to solve the unavailable problem of storage service on account of the Master node fault in huge cloud storage system, an analysis model for fault effect of management node has been constructed, which takes storage service availability, data reliability and data availability as the analysis target. Three-dimensional element has been employed in the analysis model such as fault status, real-time status and restrictive condition so as to provide an effective method for fault recovery. Based on the analysis model, a dynamic switching algorithm for master node based on message called DSA-M has been presented, in which it implements the dynamic application and switching of Master node by PRI policy based on sequence number and ensures the high availability. Test results show that DSA-M has provided management nodes auto switching and taken over while master node is breakdown and high storage service availability. The performance of DRA-M also can be stable in relative region by reasonable control of fault detection cycle and DRA-M also has strong adaptability for crush moment.

Key words: cloud storage system; Master node; fault detection; metadata; dynamic switching

1 概 述

大规模云存储系统(Huge Cloud Storage System, HCSS)凭借规模大、覆盖范围广、访问量大、存储数据量大等特点,逐步成为业界的研究热点[1],而如何保证云存储系统的高可靠性、高可用性以及故障恢复问题一直是 HCSS 的重要研究方向。HCSS 一般由一个 Master 节点,多个元数据管理节点和若干个存储节点构成,其中 Master 节点主要负责收集、检测各个元数据管理节点的工作信息,掌握云存储系统的全局运行状态。因此,当 Master 节点出现故障时,如何保证云存储服务的高可用性并尽快使其他元数据管理节点实现从普通管理节点至 Master 节点的无缝切换,是 HCSS 中需要解决的关键问题之一。

文献[2]给出了一种提高云存储系统可靠性的方法,并提出了多个云存储管理节点相互协作提供存储服务高可用和高可靠性的概念;文献[3]研究了 OpenStack 类云存储服务的同步瓶颈问题,并给出了一种轻量级的对象同步协议 LightSync,减轻了同步负载;文

收稿日期:2016-05-09 **修回日期**:2016-08-10 **网络出版时间**:2017-07-05
基金项目:国家自然科学基金资助项目(61371119)
作者简介:马玮骏(1980-),男,讲师,博士,研究方向为网络管理、网格计算、分布式存储。

献[4]描述了 GFS 通过 Master 节点 MASTER 状态、操作记录、检查点多机备份的方式确保可靠性和故障恢复的相关方法;文献[5]提出了一种基于纠删码的用于 Hadoop 平台的高性价比的容错策略,能够使用较低的代价提供云存储的高可靠性;文献[6]提出了一种基于低密度奇偶校验码的云存储系统框架,其中使用了一种裁剪纠错码来提高云存储系统的编码和解码性能;文献[7]从资源分配角度建立了云存储系统可靠性分析模型,并验证了模型的有效性;文献[8]基于云存储节点数据访问频率,提出一种副本备份算法,提高了数据的可靠性和云存储访问性能;文献[9]针对云存储系统的可靠性评价机制进行研究,并给出了面向传统被动容错和新型主动容错两类云存储系统的可靠性评价模型。为了提高云存储服务的可用性和容灾性,近年来一些研究人员将 P2P 技术融合至云存储架构[10-12]。文献[13]通过结合(n,k)-RS 编码和 X 编码,为云存储系统设计一类新的准确修复编码,在一个或两个节点发生故障时,修复局部性以及修复带宽上都具有显著优势;文献[14]利用数据拆分及编解码的思想来解决云盘存储数据的可靠性问题;文献[15]采用双活容灾存储技术,构建了一个真正意义上的双活云计算数据中心;文献[16]提出了云存储系统故障自动化管理的策略框架、策略描述语言以及策略映射机制,提高了云存储系统故障自动化管理的可实现性。

可以看出,各类云存储系统或多或少都采用了数据冗余来保证数据的可用性和可靠性,数据可靠性指数据能够持久存储并且不丢失的概率,数据可用性是指数据持续可用的概率。数据可靠性可以看作是静态需求,而数据可用性则是动态需求。数据可靠性可以由数据冗余的方式提供保证,很明显,数据可靠,却不一定可用,如果存储系统软硬件出现故障,即便数据被可靠地存储,却不能被用户使用。

因此,云存储系统必须能通过有效的软硬件动态恢复技术使系统发生故障时能够自动恢复软、硬件的正常运行,尽可能地提供持续、正确的存储服务,保证系统存储服务的高可用性,这样才能保证数据的高可用性。

为此,围绕 HCSS 中 Master 节点故障失效时持续提供云存储服务的问题,提出了 HCSS 管理节点(包括 Master 节点和元数据管理节点)故障分析模型,并以该模型为基础提出 Master 节点故障自我恢复的相关算法。

2 管理节点故障分析模型

HCSS 管理节点主要负责为云存储客户端提供元数据服务,同时需要检测各个存储节点的工作状态。通常 HCSS 中有多个管理节点,其中 Master 节点负责所有管理节点的故障检测。使用故障管理对象、故障的限制条件和表现以及故障管理对象的状态三个维度来表示 HCSS 中管理节点的故障分析模型。

设 HCSS 管理节点的故障状态空间 $S_{\mathrm{CSSM}}=\{s_1,s_2,\cdots,s_n\}$;HCSS 管理节点故障的限制条件和表现空间 $C_{\mathrm{CSSM}}=\{c_1,c_2,\cdots,c_m\}$;HCSS 管理节点实时状态空间 $R_{\mathrm{CSSM}}=\{r_1,r_2,\cdots,r_l\}$ 。在 t 时刻,存储服务可用性为 $\mathrm{SA}(t)$;数据可靠性为 $\mathrm{DR}(t)$;数据可用性为 $\mathrm{AD}(t)$ 。

记故障发生时间为 t_1,故障状态为 s_{m} ,HCSS 管理节点为 o_i ,HCSS 管理节点实时状态为 r_j ,HCSS 管理节点故障的限制条件和表现为 c_k ,故障持续时间为 Δt ,则故障 s_{m} 对存储服务可用性的影响记为:

$$F_{s_m\to\mathrm{SA}}=f_{s_m\to\mathrm{SA}}(o_i,r_j,c_k)\in[0,1] \tag{1}$$

设 $t_2>t_1$,$t_2\in(t_1,t_1+\Delta t]$,则 t_2 时刻系统存储服务可用性记为:

$$\mathrm{SA}(t_2)=\mathrm{SA}(t_1)(1-F_{s_m\to\mathrm{SA}}) \tag{2}$$

式(1)中,$f_{s_m\to\mathrm{SA}}$ 表示故障 s_{m} 对存储服务可用性的影响函数,该函数的取值范围为[0,1],取 0 表示没有影响,取 1 表示影响最大,使得系统存储服务可用性为 0,即无法提供存储服务。

同理,设 $f_{s_m\to\mathrm{DR}}$ 和 $f_{s_m\to\mathrm{AD}}$ 分别表示故障 s_{m} 对系统数据可靠性和数据可用性的影响函数,则 t_2 时刻系统数据可靠性记为:

$$\mathrm{DR}(t_2)=\mathrm{DR}(t_1)(1-F_{s_m\to\mathrm{DR}}) \tag{3}$$

t_2 时刻系统数据可用性记为:

$$\mathrm{AD}(t_2)=\mathrm{AD}(t_1)(1-F_{s_m\to\mathrm{AD}}) \tag{4}$$

为了讨论简便,下面对影响函数进行二值化:

$$F_{s_m\to\mathrm{SA}}=f_{s_m\to\mathrm{SA}}(o_i,r_j,c_k)\in\{0,1\} \tag{5}$$

$$F_{s_m\to\mathrm{DR}}=f_{s_m\to\mathrm{DR}}(o_i,r_j,c_k)\in\{0,1\} \tag{6}$$

$$F_{s_m\to\mathrm{AD}}=f_{s_m\to\mathrm{AD}}(o_i,r_j,c_k)\in\{0,1\} \tag{7}$$

式(5)~(7)表示将故障影响简化为 0 和 1 两种取值,0 表示没有影响,1 表示有影响,这种简化对于界定故障的检测范围、恢复范围以及恢复时效是很有意义的。对于故障影响为 0 的故障,可以检测但不一定要及时恢复;对于故障影响为 1 的故障,是必须要进行检测和及时恢复的。

表 1 给出了 HCSS 管理节点在崩溃故障情况下的故障分析。其中,崩溃故障指节点宕机、掉电或系统崩溃等,对 HCSS 管理节点上存储的元数据随节点崩溃的丢失情况不做任何假设。

通过表 1 可以看出,与 Master 节点紧密相关的是 r_5 状态,首先影响的是系统的存储服务可用性以及数据可用性,Master 节点崩溃会导致其他 HCSS 管理节点无法获取全局的状态信息,这必然会影响元数据管理工作,因此系统数据可靠性也会受到影响。

表 1　HCSS 管理节点崩溃故障影响分析

r_j	$F_{s_m \to SA}$	$F_{s_m \to DR}$	$F_{s_m \to AD}$
r_1 = 空闲状态	1	0	1
r_2 = 正在响应云存储客户端的读请求	1	0	1
r_3 = 正在响应云存储客户端的写请求	1	1	1
r_4 = 正在进行云存储节点的状态、故障检测	1	1	1
r_5 = 正在进行 HCSS 管理节点的状态、故障检测	1	1	1
r_6 = 正在进行 HCSS 管理节点间的元数据同步	1	1	1
r_7 = 正在响应其他 HCSS 管理节点的状态检测请求	1	0	1

对于 Master 节点故障时的存储服务可用性自恢复,关键在于故障恢复软件能否在 Master 节点发生故障时,在 r_5 状态下,使多个 HCSS 管理节点能迅速完成 Master 节点申请和接管,否则 Master 节点故障将导致系统的存储服务可用性、数据可用性都受到影响,也会影响系统的数据可靠性。

3　基于消息的 Master 节点故障动态切换算法

从故障影响分析模型的分析结果可以看出,由于同一时刻 HCSS 管理节点集合中只有一个节点作为 Master 节点,当 Master 节点出现故障,必须由其他 HCSS 管理节点动态接管 Master 节点的所有职能。

因此,提出了一种基于消息的 Master 节点动态切换算法(Dynamic Switching Algorithm for Master, DSA-M)。通过基于序号的优先级策略实现了 Master 节点失效时的 Master 节点动态申请及负载接管,保证了系统的存储服务高可用性。

3.1　DSA-M 算法设计

DSA-M 算法的核心思想是:如果某个 HCSS 管理节点 h_j 超过一定的时间没有收到 Master 节点 h_m 的检测消息,就认为 h_m 失效(崩溃),便开始申请为 Master 节点,如果所有其他节点都同意 h_j 作为 Master 节点,h_j 便接管故障检测工作。

3.1.1　超时判定策略

算法中有两个超时时限,第一个是每个 HCSS 管理节点 h_j 判断故障检测 Master 节点 h_m 失效的超时时限,该超时通过式(8)进行计算:

$$T_{me} = T + \overline{R_j} + R_j^{max} \tag{8}$$

其中,T 为故障检测周期;$\overline{R_j}$ 为 h_j 与 h_m 之间往返时延的均值;R_j^{max} 为往返时延的最大值。

如果 h_j 发现超过 T_{me} 仍然没有收到 h_m 的检测消息,则认为 h_m 出现故障;T_{me} 的选择需要谨慎,因为如果选择得过小,则会产生误判,造成频繁而没有必要的申请消息;如果选择得过大,则会造成长时间 HCSS 管理节点集合中没有 Master 节点,HCSS 管理节点的故障信息以及状态信息不能及时更新。如果出现误判,则 Master 节点返回不同意消息。

第二个超时时限是每个 HCSS 管理节点 h_j 在发出 Master 节点申请消息后,期望别的 HCSS 管理节点回执的超时时限,该超时记为 MDT,取 MDT 为当前网络往返时延的最大值。这样取值的目的一方面在于避免由于个别被申请节点的时延较大而遗漏确认消息,另一方面在于减少多个节点同时申请 Master 节点的可能性,从而减少网络通信量。

3.1.2　优先级策略

为了减少通信量,避免多个节点同时申请,为每个节点设定申请的优先级。按照优先级,当某个节点申请成功后,其他节点无需再申请。优先级采用 h_j 在云存储管理节点集合 S_{CSSMN} 中的序号 SN_j 进行定义,排在前面的节点申请优先级高,h_j 通过式(9)计算自己在怀疑 Master 节点失效后开始 Master 节点申请的等待时延:

$$T_D = (SN_j - 1) \times MDT \tag{9}$$

可以看出,排序越靠后的节点,需要等待的时间就越长。这里需要注意一个问题,序号的规定和发布是基于序号的优先级策略中的核心环节,如果序号固定,那必然存在申请不公平的现象,排序靠后的节点可能始终无法获得申请的机会。因此,算法中的序号主要通过 Master 节点进行生成和发布,而不是固定的,Master 节点只需每次检测时更新序号的排列(可以采用随机排列),便能使各个节点的优先级发生实时变化,因此 Master 节点通过序号的公平分配,即可解决申请过程中的不公平问题。

3.1.3　节点失效处理策略

(1)申请过程中被申请节点失效:申请过程中,如果某个被申请节点失效,则无法响应申请节点的申请消息,此时申请节点可以通过超时机制进行处理,一旦某个被申请节点超时,申请节点便跳过该节点,强制实行接管,这样在第一个故障检测周期内,就能够检测出该失效节点。

(2)申请过程中被申请节点超时(但没有失效):申请过程中,如果某个被申请节点由于网络或者过载等原因超时,申请节点便跳过该节点,强制实行接管,这样在第一个检测周期内,就能够检测出该节点的实时状态,对于被申请节点,一旦收到检测消息,便进行响应,并更新 Master 节点。

(3)申请过程中申请节点失效:申请过程中,如果申请节点失效,则存在两种情况。第一种是申请进行了一半,此时必定有部分被申请节点无法收到申请消息,假设没有收到申请消息的第一个节点是 h_j,则一旦到达时间 T_D,h_j 会由于没收到申请消息而发出 Master 节点申请,因此其他节点最多等待 T_D 即可收到申请消息。这里注意申请节点发送消息的顺序,应该与 SN 的顺序相反,这样,每次申请节点失效后,下一个申请节点可以在最短的时间内发出申请。例如:假设排序后的节点按 SN 顺序为 $h_1,h_2,\cdots,h_l$,申请节点为 h_1,被申请节点总数为 $l-1$,则 h_1 发送申请消息的顺序为 $h_{l-1},h_{l-2},\cdots,h_2,h_1$ 如果在发送完给 $h_k(2<k<l)$ 的申请消息后失效,则从 Master 节点失效起,经过 T_D,h_2 将发起申请,由于 $i>j$ 则 $SN_i > SN_j$,因此其余节点能在最短的时间内获得申请消息。

另一种情况是所有被申请节点都收到了申请消息后,申请节点失效,在这种情况下,由于各个节点都承认了申请节点为 Master 节点,因此可以按照针对 Master 节点的超时机制进行处理,超时时限可以按照式(8)进行设定。由于排序越靠后的节点越先得到申请消息,因此如果靠后的节点由于超时设置太小而超时的话,则会发起申请,但是该申请消息不会被其他节点所接受,于是申请节点会继续等待下一个超时周期。

3.2 DSA-M 算法描述

DSA-M 的算法描述如下所示:

算法 1:DSA-M 算法。

```
1:ON no MAIN_DETECT message from MainNode for T_me = T + R_j^ + R_j^max then// h_j
2:S_CSSMN ← S_CSSMN - h_m;SN_j ←Sequence Number of h_j in S_CSSMN ;
3:ON no MAIN_REQUEST message from any Node for T_D = (SN_j - 1) × MDT then
4:SendRequest( h_m ,MAIN_REQUEST);
5:for i = SN_l down to SN_1 do // h_i ∈ S_CSSMN = {h_1,h_2,…,h_l}
6:if i <> SN_j then SendRequest( h_i ,MAIN_REQUEST);
7:SetTimer;
8:ON ReceiveFromCSSMN( h_i ,MAIN_AFFIRMATIVE) then OK_COUNT++;
9:if( OK_COUNT= S_CSSMN . Count-1)or(Timer>MDT) then
10:SetMainNode( h_i );
11:StartDetect;
12:ON ReceiveFromCSSMN ( h_i , MAIN_NEGATIVE) then ResetToStep4;
13:ON ReceiveFromCSSMN( h_i ,MAIN_REQUEST) then
14:if SN_i ≤ SN_j then
15:SendResponse( h_i ,MAIN_AFFIRMATIVE);
16:SetMainNode( h_i );
17:ResetToStep1;
18:else SendResponse( h_i ,MAIN_NEGATIVE);
19:ON receive MAIN_DETECT message from MasterNode h_i then
20:SendResponse( h_i ,MAIN_DETECT_RESPONSE);
21:ResetToStep1;
22:ON receive MAIN_REQUEST message from other Node then//h_m
23:SendResponse( h_i ,MAIN_NEGATIVE);
```

算法 1 中假设当前 Master 节点为 h_m,当前 HCSS 管理节点为 h_j。算法将管理节点之间的消息分为五类,分别是:

(1)MAIN_DETECTMaster:节点对其余 HCSS 管理节点的故障检测消息;

(2)MAIN_DETECT_RESPONSE:HCSS 管理节点对故障检测的响应消息;

(3)MAIN_REQUEST:某个 HCSS 管理节点申请成为 Master 节点的申请消息;

(4)MAIN_AFFIRMATIVE:某个被申请节点同意申请节点的 Master 节点申请消息;

(5)MAIN_NEGATIVE:某个被申请节点不同意申请节点的 Master 节点申请消息。

第 1 行表示一旦任意一个 HCSS 管理节点 h_j(非 Master 节点)超过 T_{me} 都没有收到来自 Master 节点 h_m 的消息,则启动 2 ~ 21 行的算法任务。

第 2 行表示在 S_{CSSMN} 中去除 h_m 进行重新排序,得到每个节点的 SN。

第 3 ~ 12 行表示 h_j 超过 T_D 仍然未收到来自其他 HCSS 管理节点的申请消息 MAIN_REQUEST,而执行的算法任务。

第 4 行表示首先给 h_m 发送申请消息 MAIN_REQUEST,防止对 Master 节点的误判造成系统中存在多个 Master 节点的情况。

第 5 ~ 6 行表示 h_j 按照 SN 倒排的顺序对其他 HCSS 管理节点发出申请消息 MAIN_REQUEST。

第 7 行表示发送完消息即设定一个定时器,用于检测对申请消息响应的超时。

第 8 行表示 h_j 发出申请消息后收到了某个 HCSS 管理节点 h_i 的同意消息 MAIN_AFFIRMATIVE,此时 h_j 将计数器加 1,用于统计收到的同意消息数;第 9 ~ 11 行表示如果 h_j 收到的同意消息数等于被申请节点数,或者等待同意申请消息超时,便置自己为 Master 节点,同时开始故障检测。

第 12 行表示如果 h_j 收到某个 HCSS 管理节点 h_i 的不同意消息 MAIN_NEGATIVE,则 h_j 重置本地定时器,恢复到第 4 步的初始状态,即刚发现 h_m 失效的状态,继续等待 T_D,ResetToStep4 函数负责状态的重置工作。需要注意的是,一旦 h_j 的状态被重置,则它不会

再处理任何 MAIN_AFFIRMATIVE 或者 MAIN_NEGATIVE 消息。

第 13 行表示 h_j 收到来自 h_i 的申请消息 MAIN_REQUEST,第 14 ~ 18 行表示对消息中 h_i 的序号 SN_i 进行判断,如果 $SN_i \leqslant SN_j$,表示 h_i 满足申请条件(优先级条件),h_j 便使用同意申请消息 MAIN_AFFIRMATIVE 响应 h_i,同时置 h_i 为 Master 节点,重置自身的状态到发现 Master 节点超时时的状态,进入正常的运行流程,ResetToStep1 函数负责重置处理;如果 $SN_i > SN_j$,则表示 h_i 不满足申请条件(优先级条件),h_j 便使用不同意申请消息 MAIN_NEGATIVE 响应 h_i。

第 19 ~ 21 行表示 h_j 收到来自 Master 节点的故障检测消息,则立即响应 MAIN_DETECT_RESPONSE 消息,并重置自身的状态到发现 Master 节点超时时的状态,进入正常的运行流程。

第 22 ~ 23 行是 Master 节点 h_m 算法,这里表示如果由于网络状态不稳定或者 h_m 过载,造成 h_j 的误判,则 h_m 响应不同意申请消息 MAIN_NEGATIVE。

该算法的优点在于:

(1)无需实时获取各个 HCSS 管理节点的状态,由于 Master 节点失效,状态检测结果可能会不一致,采用序号的方式进行 Master 节点申请计算工作,能够在 HCSS 管理节点状态不一致的情况下完成申请,并正确运行;

(2)采用序号的方式进行申请优先级计算,避免了不必要和过多的申请消息,申请过程中任何 HCSS 管理节点失效,算法依然能够保证正确性;

(3)Master 节点申请和接管对于每个节点只需 1 次往返消息交互,消息代价很小,复杂度为 $O(N)$,N 为 HCSS 管理节点数;

(4)可扩展性好,当 HCSS 管理节点增多,增加的消息量为常数,与节点数目无关;

(5)正常情况下,Master 节点失效后,经过一个故障检测周期加上网络时延的时间,即可完成接管,不论网络条件如何,都能自动进行恢复,保证故障检测的有效性。

因此,DSA-M 算法在 Master 节点处于状态 h_j 并出现故障的情况下,保证了系统的数据可靠性。

4　实验及结果分析

实验采用 5 个 HCSS 管理节点,其中 5 号节点作为初始 Master 节点,在 5 个节点上运行 DSA-M 算法,测试当 Master 节点失效时,其他 HCSS 管理节点申请 Master 节点的性能。

测试中,故障检测超时取 800 ms 和 200 ms,故障检测周期为 1 000 ms,初始故障检测 Master 节点编号为 5,故障检测 Master 节点申请回执超时为 100 ms,测试进行 100 次。

测试过程中,使 HCSS 管理节点 5(故障检测 Master 节点)停止响应任何请求(模拟崩溃故障,即失效),Master 节点失效的时刻是随机的,记录 Master 节点失效的时刻 t_1 与其他 HCSS 管理节点申请 Master 节点成功的时刻 t_2 之差:$\Delta t = t_2 - t_1$(申请时间),用于衡量 Master 节点失效状态下其余节点申请 Master 节点的性能。

测试结果如图 1 所示。

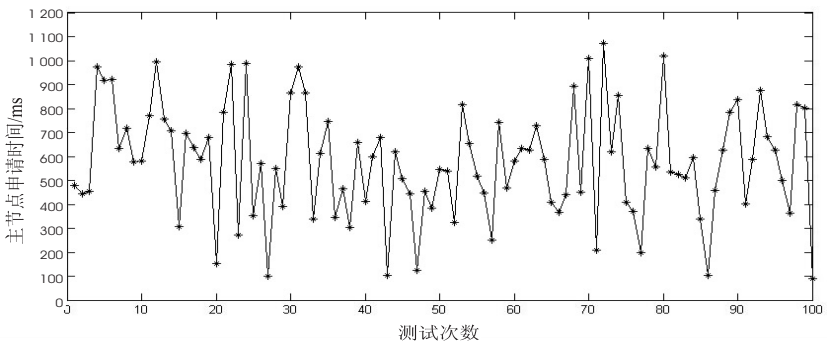

图 1　Master 节点失效时 HCSS 管理节点申请 Master 节点时间

由图 1 可知,申请时间基本上在 80 ~ 1 080 ms 内随机变化,这主要是因为故障检测周期 T 为 1 000 ms,而在故障检测周期内的任意一个时刻,Master 节点都可能失效,其余节点判断 Master 节点失效的规则是超过 T_{me} 还没有收到来自 Master 节点的故障检测消息,则认为 Master 节点失效。$\overline{R_j} + R_j^{max}$ 在测试中比较小,基本上在 50 ms 左右,而此后 1 号节点首先发起申请,并且获得其他节点的响应,其中往返时延以及多个节点的同步时间大约在 30 ms 左右。假设网络往返时延为 T_R,节点的计算时延为 T_C,Master 节点失效时刻到下一个故障检测周期的时间差(即等待检测周期到达的时间)为 T_W,$T_W < T$,则申请时间 Δt 为:

$$\Delta t = T_W + T_R + T_C + \overline{R_j} + R_j^{max} \tag{10}$$

其中,考虑到实验环境,$T_R + T_C + \overline{R_j} + R_j^{max}$ 相对比较稳定,变化不大,基本上在 100 ms 左右,因此 Δt 基本上随 T_W 在 T 的范围内变化,变化范围如下:

$$\Delta t \in (T_W^{min} + T_R + T_C + \overline{R_j} + R_j^{max}, T_W^{max} + T_R + T_C + \overline{R_j} + R_j^{max}) \quad (11)$$

由于 $T_W^{min} = 0, T_W^{max} = T$,则有:

$$\Delta t \in (T_R + T_C + \overline{R_j} + R_j^{max}, T + T_R + T_C + \overline{R_j} + R_j^{max}) \quad (12)$$

为了体现测试的全面性,考察当多个节点失效时 Master 节点申请的性能,失效节点组合按照(5)、(5,1)、(5,1,2),(5,1,2,3)四种,每一种组合内的节点都是同时失效,由 DSA-M 算法描述可知,申请的优先级是以节点序号计算,只要排序靠前的节点没有失效,排序靠后的节点将无法完成申请,因此以上四种组合使得每个节点(1,2,3,4)都有机会申请 Master 节点,是最有代表性的一种组合。

此外,由图 1 的分析可知,申请时间的变化依赖于 Master 节点失效时刻在故障检测周期内的分布,因此为了讨论方便,假设 Master 节点失效时刻按照间隔 100 ms 的规律均匀变化,考察一个检测周期内 HCSS 管理节点申请 Master 节点时间与失效节点组合之间的关系,如图 2 所示。

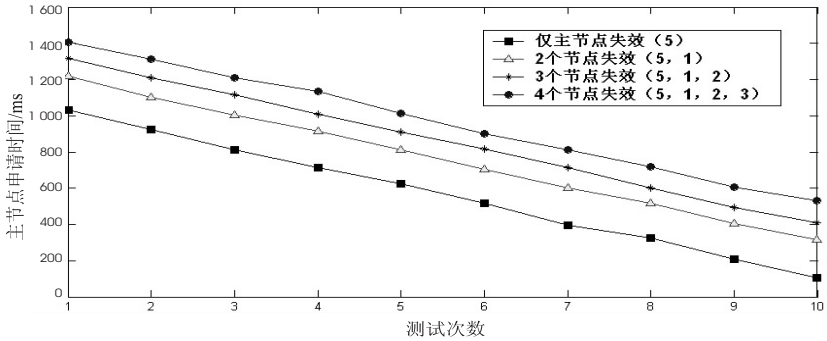

图 2 多个节点失效时申请 Master 节点成功时间(失效时刻均匀分布)

从图 2 可以看出,节点失效的不同组合所对应的 Master 节点申请时间并不相同,这种差异主要来自于 Master 节点申请优先级的控制。由 DSA-M 算法描述可知,每个节点发现 Master 节点失效后,将延迟时间 T_D 发送 Master 节点申请消息,因此排序越靠后的节点发起申请的延迟就越大,而延迟的时间和节点的序号以及最大往返时延(MDT)相关,测试中 MDT=100 ms。图 2 中 Master 节点失效后,可以申请成为 Master 节点的 HCSS 管理节点有 4 个,因此当前 n 个节点都失效时,最大等待时延 T_{wait}(申请开始时间)和申请完成时间 T_{finish} 分别为:

$$T_{wait} = n \times MDT \quad (13)$$

$$T_{finish} = MDT + T + T_R + T_C + \overline{R_j} + R_j^{max} \quad (14)$$

式(14)因为有多个节点失效,必然有节点会响应申请消息超时,因此会多一个超时时间 MDT,这也是图 2 中多于 1 个节点失效时的申请时间要比只有 Master 节点失效时的申请时间普遍多 100 ms 左右的原因。因此在多个节点失效的情况下,申请时间可以使用式(15)计算。

$$\Delta t = (n + 1) \times MDT + T_W + T_R + T_C + \overline{R_j} + R_j^{max} \quad (15)$$

其中,n 表示从 1 号节点开始,连续失效的节点个数。

因此在多个节点失效的情况下,申请时间的变化范围为:

$$\Delta t \in (T_W + T_R + T_C + \overline{R_j} + R_j^{max}, (n + 1) \times MDT + T_W^{max} + T_R + T_C + \overline{R_j} + R_j^{max}) \quad (16)$$

可以看出,通过控制故障检测周期 T,便能够使得 DSA-M 算法的性能保持在相对稳定的区间内,随失效时刻的适应性也比较强。

5 结束语

存储服务可用性是衡量云存储系统好坏的重要指标之一,当云存储系统中管理节点出现故障时,如何快速恢复系统运行,确保存储服务的可用性是 HCSS 需要解决的关键问题之一。针对 HCSS 中 Master 节点失效导致存储服务不可用的问题,建立了 HCSS 中管理节点故障影响分析模型,根据模型分析结果,提出了 DSA-M 算法。当 Master 节点发生故障时,以多个 HCSS 管理节点之间并行工作为基础,通过基于序号的优先级策略实现了 Master 节点的动态申请和切换,保证了云存储服务的高可用性。

测试结果表明,DSA-M 算法能够在 Master 节点发生故障时自动进行 Master 节点切换和接管,恢复云存储服务的运行。通过控制故障检测周期,能够使 DSA-M 算法的性能保持在相对稳定的区间内,随失效时刻的适应性也比较强。

参考文献:

[1] 刘 鹏. 云计算[M]. 第 3 版. 北京:电子工业出版社,2015.

[2] 马玮骏,吴海佳,刘 鹏. MassCloud 云存储系统构架及可靠性机制[J]. 河海大学学报:自然科学版,2011,39(3):

348-354.
[3] Chekam T T, Zhai E, Li Z, et al. On the synchronization bottleneck of OpenStack swift-like cloud storage systems[C]// IEEE international conference on computer communications. San Francisco: IEEE, 2016: 135-144.
[4] Kamath A, Jaiswal A, Dive K. From idea to reality: Google file system[J]. International Journal of Computer Applications, 2014, 103(9): 8-10.
[5] Wu C H, Hsu P. Cost-effective and reliable cloud storage for big data[C]//ASE big data & social informatics 2015. New York: ACM, 2015.
[6] Wei Y, Yong W F. A cost-effective and reliable cloud storage [C]//2014 IEEE 7th international conference on cloud computing. Anchorage: IEEE, 2014: 938-939.
[7] Faragardi H R, Shojaee R, Tabani H, et al. An analytical model to evaluate reliability of cloud computing systems in the presence of QoS requirements[C]//International conference on computer and information science. Niigata, Japan: [s. n.], 2013: 315-321.
[8] Joolahluk J, Wen Y F. Reliable and available data replication planning for cloud storage[C]//IEEE international conference on advanced information networking and applications. Barcelona, Spain: IEEE, 2013: 772-779.
[9] 赵　畅. 云存储系统可靠性分析[D]. 天津: 南开大学, 2014.
[10] Song J, Deng J H. NOVA: a P2P-cloud VoD system for IPTV with collaborative pre-deployment module based on recommendation scheme[C]//International conference on materials science and information technology. Nanjing, China: [s. n.], 2013: 1566-1570.
[11] Chakareski J. Cost and profit driven cloud-P2P interaction [J]. Peer-to-Peer Networking and Applications, 2015, 8(2): 244-259.
[12] Babaolu O, Marzolla M, Trambrini M. Design and implementation of a P2P cloud system[C]//Proceedings of the ACM symposium on applied computing. Trento, Italy: ACM, 2012: 412-417.
[13] 李小兵, 许胤龙, 林一施, 等. X 再生码: 一类适用于云存储的准确修复编码[J]. 计算机应用与软件, 2014, 31(8): 241-244.
[14] 王　帅, 常朝稳, 王禹同, 等. 面向多云盘的 N+i 模式的编码冗余存储机制[J]. 小型微型计算机系统, 2016, 37(2): 236-240.
[15] 吴礼乐. 基于双活容灾存储技术的云计算数据中心的设计及应用[J]. 电子设计工程, 2015, 23(6): 190-192.
[16] 马玮骏, 王　强, 何晓晖, 等. 一种基于策略的云存储系统故障管理方法[J]. 软件工程, 2016, 19(6): 4-7.

(上接第 84 页)

结合路径权重,基于知识地图技术的企业知识推荐算法。即根据知识地图模型利用知识地图节点间的权重关系,得到相似度更高的用户预测集合,并以此来向目标用户推荐知识。理论分析表明,改进算法有效解决了企业知识推荐匹配程度低的问题,可以准确获取用户之间的关联度,提高推荐算法的质量和个性化程度。但该算法尚未考虑项目间的语义关系和用户间的语义关系,一定程度上影响了相似度计算和推荐效果,需要今后进一步研究解决。

参考文献:

[1] 朱文奇. 推荐系统用户相似度计算方法研究[D]. 重庆: 重庆大学, 2014.
[2] Chung W, Chen H, Nunamaker J F. A visual framework for knowledge discovery on the web: an empirical study of business intelligence exploration[J]. Journal of Management Information Systems, 2005, 21(4): 57-84.
[3] Yoon B, Lee S, Lee G. Development and application of a keyword based knowledge map for effective R&D planning[J]. Scientometrics, 2010, 85(3): 803-820.
[4] Gordon J L. Creating knowledge maps by exploiting dependent relationships[J]. Knowledge-Based Systems, 2000, 13(2): 71-79.
[5] Selvi R T, Prakash-Raj G D. Information retrieval models: a survey[J]. International Journal of Research and Reviews in Information Sciences, 2012, 2(3): 227-233.
[6] 刘　刚. 面向领域的软件需求一致性验证方法研究[D]. 哈尔滨: 哈尔滨工程大学, 2008.
[7] 程　飞, 贾彩燕. 一种基于用户相似性的协同过滤推荐算法[J]. 计算机工程与科学, 2013, 35(5): 161-165.
[8] 王有远, 赵　璐. 面向产品设计的多维度知识推送研究[J]. 制造业自动化, 2015, 37(14): 131-133.
[9] 蒋翠清, 李斌生, 高家飞, 等. 面向协同的产品设计知识推送研究[J]. 中国机械工程, 2012, 23(16): 1972-1977.
[10] 王晓堤, 桑　婧. 基于云模型的时间修正协同过滤推荐算法[J]. 计算机工程与科学, 2012, 34(12): 160-163.
[11] 张　艳, 梁欣欣, 张耐民, 等. 基于知识地图的航天知识推送方法研究[J]. 航天工业管理, 2015(5): 35-37.
[12] 唐积益, 黄树成. 优化相似度计算在推荐系统中的应用[J]. 电子设计工程, 2015, 23(23): 46-48.
[13] 刘青文. 基于协同过滤的推荐算法研究[D]. 合肥: 中国科学技术大学, 2013.
[14] 熊　奇. 基于知识地图的知识检索与推荐方法研究[D]. 上海: 上海交通大学, 2009.