文章编号: 1000-5641(2016)05-0027-09

分布式内存数据库系统的容错管理

赵镇辉, 黄承晟, 周敏奇, 周傲英

(华东师范大学 数据科学与工程研究院, 上海 200062)

摘要: 在大数据背景下, 分布式系统被企业广泛部署和应用, 随着分布式系统节点规模的扩大, 系统故障的概率也将随之增加, 在分布式系统中引入容错机制, 对提升分布式系统可用性、可靠性、可恢复性至关重要. CLAIMS 系统是面向金融领域的对实时数据进行实时分析的内存数据库系统——在数据不断注入系统时, 提供近实时的查询、分析任务. 本文主要探讨 CLAIMS 系统中容错机制. 依据租约机制, 实现系统中异常节点的快速发现及标记(即 Fail-fast). 在标记异常节点之后, 实现对受影响分析任务的重启(即 Fail-over); 对异常节点全局内存状态的恢复(即 Fail-back). 实验结果表明, 本文所提算法能够较好地实现 CLAIMS 系统的容错特性.

关键词: 分布式内存数据库; 容错; 租约

中图分类号: TP392 **文献标识码**: A **DOI:** 10.3969/j.issn.1000-5641.2016.05.004

Fault-tolerance in distributed in-memory database systems

ZHAO Zhen-hui, HUANG Cheng-shen, ZHOU Min-qi, ZHOU Ao-ying

(*Institute for Data Science and Engineering, East China Normal University, Shanghai* 200062, *China*)

Abstract: In the big data era, distributed system has been widely deployed and applied in various fields. Nevertheless, the more nodes involved, the higher probability of system failures may occur. It is important to introduce fault-tolerance mechanism for distributed systems to achieve even higher performance, higher reliability and higher availability. CLAIMS system is an in-memory database system for real-time data analysis, which is mainly used for financial applications. It provides near real time query task and analytic task. This paper mainly discuss fault-tolerance mechanism in CLAIMS. Achieve lease-based quick system failure detection (Fail-fast). Achieve restart of affected analytic task after detecting failure (Fail-over). Achieve in-memory state recovery of abnormal node. Experiment indicate that the algorithm presented in this paper can achieve fault-tolerance in CLAIMS.

Key words: distributed in-memory database; fault-tolerance; lease

---

收稿日期: 2016-06

基金项目: 国家自然科学基金重点项目(61332006); 上海市基金(13ZR1413200)

第一作者: 赵镇辉, 男, 硕士研究生, 研究方向为分布式数据库.

通信作者: 周敏奇, 男, 教授, 研究方向为对等计算、云计算、分布式数据管理和内存数据管理系统.
E-mail: mgzhou@sei.ecnu.edu.cn.

0 引 言

在大数据环境下, 大型互联网公司对高性能海量数据处理的需求大幅增加. 在廉价 PC 服务器上部署的分布式数据库系统能进一步降低数据处理的成本, 同时获得数据处理的高吞吐率, 高可用性, 高可靠性, 分布式系统成为处理高性能海量数据的首选. 目前, 阿里旗下的公司蚂蚁金服及阿里巴巴自主研发的通用关系数据库 OceanBase 已经支撑淘宝、天猫和聚划算的所有日常交易. 分布式数据库 OceanBase 具有自动检测服务器故障检测与容错的功能. 2015 年“双十一”阿里旗下的天猫商城全天成交金额为 912.17 亿元, 订单数达到了 4.67 亿, 开场1分12秒后就达到了成交金额就达到了 10 亿元. 在服务器如此高负载的情况下, 系统的容错显得格外重要, 避免如订单失效、数据丢失、查询错误等问题. 由 Postgres 和 Ingres 联合创始人 Mike Stonebraker 开发的内存数据库 VoltDB, 使用K-safety机制来保证数据的安全. 而所有容错机制对于用户来说是不可见得[1]. 这也是容错的另一大特性.

分布式系统容错分为两个阶段, 错误检测和错误处理. 错误处理阶段又可分为两种策略 Fail-over 与 Fail-back, 前者是通过转移失效机器上未完成的任务来实现查询处理复杂的容错, 后者通过失效机器重新激活后或者新机器替换失效机器后恢复失效机器的全局状态来保证数据一致性[2].

结合目前CLAIMS分布式数据库系统的架构, 本文给出了针对不同情况的容错机制, 具体如下.

(1) 在 CLAIMS 系统中增加基于租约机制的容错检测系统.

(2) 在 CLAIMS 系统中实现基于容错检测机制后的 Fail-fast 机制来及时发现节点失效.

(3) 在 CLAIMS 系统中实现基于容错检测机制的 Fail-over 机制来保证部分节点宕机时 CLAIMS 系统能够将查询负载重新分发到其他正常节点并继续提供服务.

(4) 在 CLAIMS 系统中实现了基于容错检测机制的 Fail-back 机制来保证节点恢复过程中 CLAIMS 中的节点重新加入集群中, 并继续执行计算.

本文的内容组织: 第 1 节介绍背景知识; 第 2 节介绍预备知识; 第 3 节介绍容错算法; 第 4 节评估实验; 第 5 节总结全文.

1 相关工作

1.1 CLAIMS 系统介绍

系统架构介绍分为外部架构与内部架构, 外部架构表现客户与 CLAIMS 系统的关系. 内部架构则分析了 CLAIMS 系统的主节点与从节点内部的主要环境.

1.1.1 外部架构

CLAIMS 是一个开源的分布式内存数据库系统. 通过具有高吞吐实时数据注入的功能实现了实时数据分析. 系统在处理 SQL 时, 所有数据和中间结果都存于内存中, 避免了磁盘的 I/O 开销, 实现了高吞吐量情况下高效的数据分析性能. 用户使用在外部主机上运行的 Client 端, 输入 SQL 语句. CLAIMS 外部架构图见图 1.

1.1.2 内部架构

CLAIMS 的内部结构详见图 2, 主节点包括 SQL 解析器、查询优化器、数据字典管理器、资源管理器、调度器和存储管理器. 主节点接收 Client 端发来的 Sql 语句并对 SQL 请求进行解析与查询优化, 将查询计划派发到不同的从节点上. 从节点结构包括了执行器、数

字字典管理器、资源管理器、调度器和存储管理器. 从节点执行主节点发送的物理查询计划, 同时负责底层的文件系统进行数据的存储与接收[3].

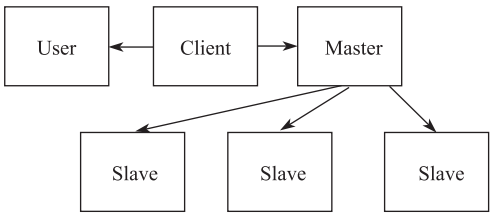

图 1　CLAIMS外部架构图

Fig. 1　CLAIMS external architecture diagram

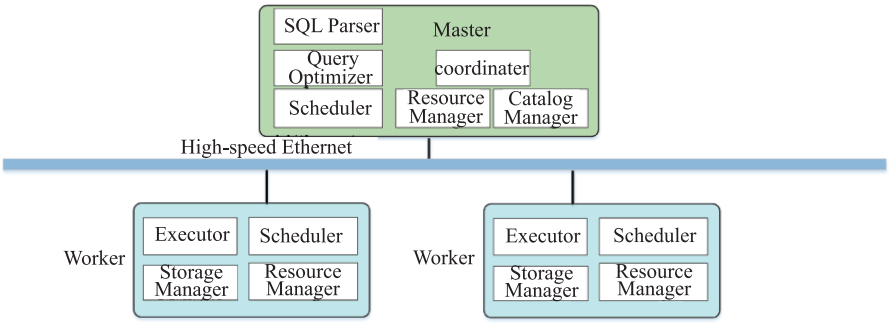

图 2　CLAIMS内部架构图

Fig. 2　CLAIMS internal architecture diagram

1.1.3　问题阐述

CLAIMS 系统是运行在较大数据集上的实时数据分析系统. 实时数据分析要求 CLAIMS 系统容错机制的时间开销很小, 在一次数据的实时数据分析中, 节点的非拜占庭错误不会使用户获得错误结果, 所造成的额外的时间开销也应该被控制. 系统在检测到系统中存在节点失效时, 会返回”错误”并重启失效节点.

1.2　预备知识

CLAIMS 中的容错是基于租约机制的实现, 租约为其提供了理论的基础. 在实现中, 本系统采用 CAF 框架, CAF 为 CLAIMS 提供了多线程高性能网络通信库. 结构图见图 3.

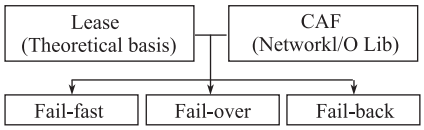

图 3　CLAIMS 容错结构图

Fig. 3　CLAIMS fault-tolerance structure diagram

1.2.1　租约介绍

1989 年斯坦福大学的 Gray C 和 Cheriton D 提出了利用租约来维护缓存一致性的方法[4]. 租约是指服务器给予客户端在一定期限内可以控制读写操作的权利, 当服务器试图修改数据时, 首先向拥有这块数据的租约的客户端发送请求. 客户端从服务器读取数据时就同时获取租约, 如果在租约期限内, 没有收到服务器的修改请求, 就可以保证当前缓存中的内容是最新的. 租约过期后, 如果客户端还需要读取数据, 则必须重新获取租约即“续约”. 租约分为短租约与长租约. 短租约维护开销较大, 一般短租约时间长度为秒级别的, 而长租约续

约的开销会小很多. 在CLAIMS中结合短租约实现了基于租约的心跳机制.

1.2.2 CAF Actor 模型框架介绍

CLAIMS 系统中, 使用 CAF 完成节点间的通信, 实现 Fail-fast 机制. CAF 是一个轻量级通信框架. CAF 的实现方法是, 在线程的级别上再创建一个 Actor 结构, Actor 承担原来系统结构中线程的角色, 然后线程池的线程轮转完成 Actor 的指令. 使用这种结构, 创建2^{20}个 Actor 在4~64 核的机器上时间开销小于 2 s, 性能比同类型其他 Actor 框架高很多. 在消息传输上, 在 64 核的机器上, 100 个 Actor 对 1 个 Actor 各发送 1 000 000 条消息(总共 100 000 000 条消息)用时为 86 s, 同样的任务在使用 Scala 的通信框架时则需花费 1 086 s[5]. 此外, CAF 还支持无锁编程, 并提供了错误检测, 在 CAF 中消息只有发送成功与发送失败两种状态, 编程者不需要考虑重发, 跨平台等问题.

2 算法介绍

主要介绍 3 种算法, Fail-fast 实现错误检测, Fail-over 实现集群失效时继续为外界提供服务, Fail-back 实现集群中节点重启恢复后再次提供服务[6].

2.1 Fail-fast 算法

CLAIMS中使用短租约即心跳机制主动去监测节点宕机或网络拥塞所造成的节点失效. 主节点(Master)启动一个线程级别的 Master Actor 去监听某一个端口并接受集群其他节点的心跳和其他消息. 从节点将向主节点的 Master Actor 发送注册请求, 主节点检查从节点的合法性(检查是否包含重复 IP 和端口, 假设集群中的所有节点都不会发送恶意的信息.)后分配给从节点一个全局唯一的节点 ID. Master Actor 将该从节点 ID 加入到由自身维护的存活列表 L 中, 并且在一定周期 T_1 内增加列表 L 中的每个节点i的心跳计数C_i, 当C_i达到$C_{\max}$时将 L 中的该节点标记为死亡[7]. 当从节点i收到注册成功请求后表示注册成功, 将在一个周期 T_2 内发送心跳信息给主节点. Master Actor 接收到心跳信息后就会将该节点对应的C_i清 0, 并且返回当前存活节点的所有信息, 使从节点获得当前系统中的存活节点信息, 这些信息在多 Master 架构或多 Coordinator 架构中是至关重要的[8]. 原理图见图 4.

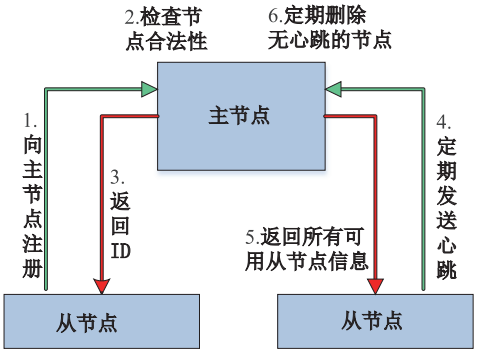

图 4 CLAIMS 系统 Fail-fast 算法原理图

Fig. 4 The principle of Fail-fast algorithm in CLAIMS system diagram

2.2 Fail-over算法

Fail-over 是一种容错机制, 在分布式系统中的概念是越过失败并继续向用户提供服务[9]. 在 Claims 中 Fail-over 将保证在节点出现故障时, 不中断地对外服务. 主节点(master)上的协调器保持监听从节点(slave)的活跃信息(心跳机制), 当某个节点不可用(丢失或死亡)时, 协调器(Coordinator)将节点的死亡情况标识给资源管理器(Resource Manager), 资源管理器

将获得这个节点的死亡信息, 并将其标记为死亡节点. 同时, master 会终止该从节点上所有未执行的 query 的租约的“续租”, 所有的从节点在执行这些 query 的计算都将因“续租”而停止运算, 清空相关中间数据.

2.3　Fail-back 算法

Fail-back 机制, 即当一台机器宕机后, 机器能够恢复到正常的状态, 继续工作[10]. 恢复的主要目的是恢复原来该机器内存中的数据, 以及已持久化的数据. 当机器未宕机, 只是从节点上的程序崩溃时, 在主节点(Master) fail-fast 机制中会发现节点失效, 发送重启命令, 启动存放在指定目录下的重启脚本, 重启该 slave 程序. 当机器出现宕机或丢失等情况时, Master 在n次尝试无效后, CLAIMS 系统会发出警报通知管理员, 集群管理员重启机器. 原理图见图 6.

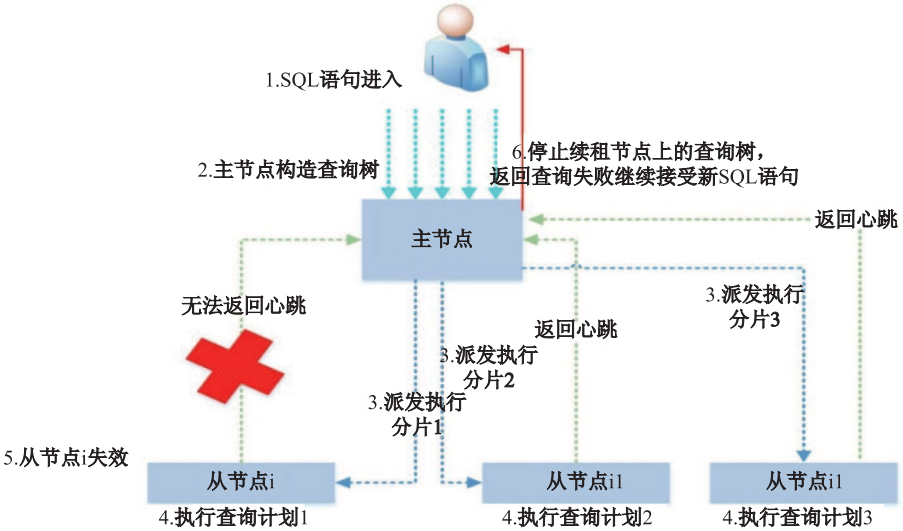

图 5　CLAIMS 系统 Fail-over 算法原理图

Fig. 5　The principle of Fail-over algorithm in CLAIMS system diagram

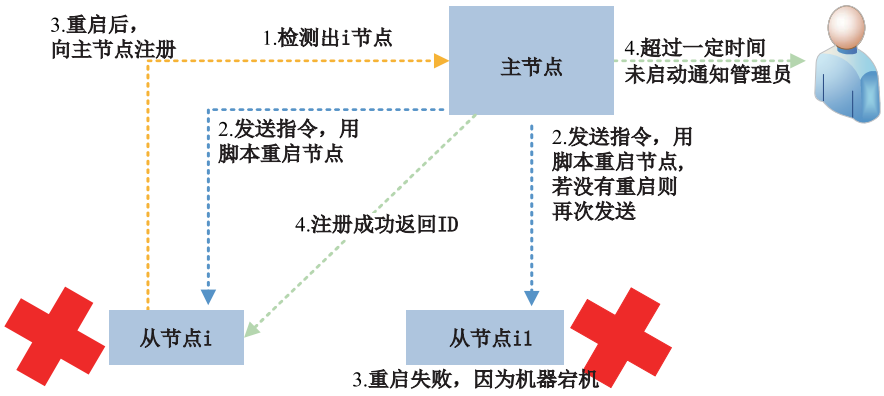

图 6　CLAIMS系统 Fail-back 算法原理图

Fig. 6　The principle of Fail-back algorithm in CLAIMS system diagram

3　实　　验

3.1　实验环境

实验由 3 部分组成, 分别为 Fail-fast, Fail-over, Fail-back. 实验运行于一组 3 台 PC 组

成的分布式集群中, 其中一台作为 Master 节点两台(a、b)作为 Slave 节点. 硬件环境均为i7-4790 3.6Ghz*8、同批次 1 t机械硬盘.

实验中使用 TPC-H 1G 数据集作为测试数据集.

3.2 Fail-fast 实验

实验内容

在上述实验环境中部署分布式的 CLAIMS 系统, 分别为 Master 和 Slave-a, Slave-b. 在 CLAIMS 系统正常运行后, 手动停止 Slave-a, 分别统计不同心跳间隔时, Master 节点检测到子节点 Slave-a 丢失的平均时间. 通过遍历存活列表, n次未发现节点存活信息则认为此节点丢失.

分别设定心跳间隔为 1、1.5、2、2.5 和 3, 分别设定n为 3、5, 共 10 种状态, 每种状态运行 5 次并计算平均时间.

实验结果

见图 7. MaxTry 表示为最大尝试次数, 当超过尝试次数 Master 会将该 Slave 标记为死亡的.

X 轴 Frequency 表示为每次发送心跳的周期长短, 1.5 表示 1.5 s 发送一次心跳, Master 也会在每个 1.5 s 更新一次自己的存活列表. Y轴 time 表示发现死亡的时间, 如 MaxTry=3 时, Frequency= 1.5 s/times 时需要 4.5 s 左右发现节点失效.

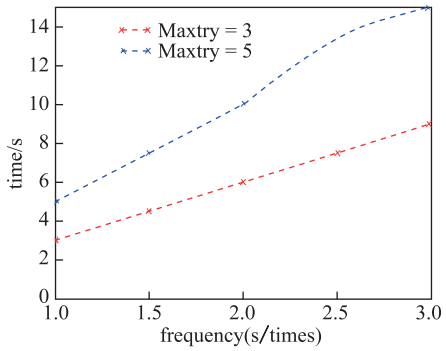

图 7 检测到心跳时间与频率图

Fig. 7 Check time with different heartbeat timeout and frequency

综合图中显示的实验结果可以看出, 发现错误的时间与搜索间隔基本呈线性增长, 更长的心跳间隔会产生更长的错误响应时间. 较短的心跳间隔会产生网络阻塞而导致的错误报警, 从而造成额外的容错开销, 所以根据实际应用场景选择适合的心跳周期, 是非常重要的.

在 CLAIMS 系统中, MaxTry 设置为 3, 心跳间隔设为 1 s, 由在主节点上的协调器负责收集其他从节点上的心跳信息, 当 3 次没有收到从节点的心跳就将从节点标记为死亡.

3.3 Fail-over 实验

实验设计思路及内容

在 CLAIMS 系统中完成一组多个 SQL 语句的查询计划, 并且在固定语句时使节点丢失, 记录所有的查询时间, 并且对比在未发生节点丢失的情况下的查询时间, 通过对比查询时间的差异, 来判断 Fail-over 模块的实际运行情况. 步骤如下.

(1) 启动 CLAIMS 系统, 导入 1G TPC-H数据集.

(2) 输入 TPC-H SQL 开始进行查询(共计 8 条, 依次输入).

(3) 手动停止 Slave-a 上的 CLAIMS 进程(在执行 TPC-H6 时).

(4) 获取结果(查询时间).

(5) 重复 2—4 步骤 6 次, 当偶数次时, 不进行第 3 步, 统计所有的结果.

通过对比结果时间, 可以看出, 除去第 4 组测试语句, 大部分语句的运行时间相差很小, 在奇数次实验时, 第 4 个 SQL 查询比偶数次实验时耗时短很多, 并在其后的语句中, 用时略微超过偶数次实验. 在第 4 个手动中断 slave-a 上的 CLAIMS 进程, 导致系统直接返回错误, 语句未完成便返回错误. 除此之外, 奇数次实验的 4～8 个 SQL 语句的查询耗时均比偶数次实验有小幅度的增加, 在有节点故障后, 系统性能略有下降, fail-over 模块正常工作, 并达到预期设计效果.

在 CLAIMS 系统中, 发现有节点失效的信息, 立刻停止在该节点上所分配任务的租约续租并且返回给客户端错误信息, 每个从节点在执行计划时会定期向主节点续租, 当得到无法续租的信息后, 将放弃执行该语句.

表 1　CLAIMS 系统完成各 TPC-H SQL 时间

Tab. 1　Time result for different TPC-H SQL in CLAIMS

测试规范\实验	第1次/s	第2次/s	第3次/s	第4次/s	第5次/s	第6次/s
TPC-H1	5.5	5.4	5.5	5.5	5.5	5.5
TPC-H3	1	1.1	1	1	1.1	1.1
TPC-H5	10.7	10.7	10.8	10.8	10.8	10.8
TPC-H6	3	4.4	3	4.4	3.1	4.4
TPC-H10	4.4	4.3	4.4	4.3	4.4	4.3
TPC-H12	1	0.8	0.9	0.9	1	0.9
TPC-H13	0.9	0.7	0.9	0.7	0.8	0.7
TPC-H17	4.2	4	4.2	4.1	4.1	4.1

3.4　Fail-back 实验

实验设计思路及内容

模拟子节点宕机情况, 通过检查主节点上的存活列表, 通过子节点是否重新出现在存活列表上, 判断子节点是否被成功 Fail-back.

同 Fail-fast 实验, 使 CLAIMS 系统运行在 3 台机器组成的集群上, 使一台 Slave 所在机器宕机, 记录在节点宕机后, 在一定时间内 Master 节点上存活列表中存活节点个数.

实验结果表如下(见表 2 和 3).

表 2　存活节点与时间图(心跳频率为 1 s/次)

Tab. 2　Number of alive node with times (frequency=1 s/times)

	第1秒	第2秒	第3秒	第4秒	第5秒	第6秒
M3F1	2	2	2	1	2	2
M5F1	2	2	2	2	1	2

表 3　存活节点与时间图(心跳频率为 0.5 s/次)

Tab. 3　Number of alive node with time (frequency=0.5 s/times)

	第0.5秒	第1秒	第1.5秒	第2秒	第2.5秒	第3秒
M3F0.5	2	2	1	2	2	2
M5F0.5	2	2	2	2	1	2

实验结果分析

表2记录的是心跳周期为 1 s, 检测间隔是 1 s, 最大尝试次数为 3 次和 5 次的实验结果情况, 表3记录的心跳周期为0.5 s, 检测间隔是 0.5 最大尝试次数为 3 次和 5 次的实验结果.

在 MaxTry=3, Frequency=1 s/times 时, 在 4 s 时, 发现存活列表中只有 1 个节点存活, 在第 5 s 是, 存活节点个数增加为 2. 在 MaxTry=5, Frequency=1 s/times 时, 在 5 s 存活节点减少为 1, 并且在第 6 s 时重新加入存活列表.

在 MaxTry=3, Frequency=0.5 s/times 时, 在 1.5 s 左右发现了节点丢失, 并且在 2 s 左右重新加入存活列表, 在 MaxTry=5, Frequency=0.5 s/times 时, 在 2.5 s 左右发现了节点丢失, 并且在 3 s 左右重新加入存活列表.

综合以上结果, 更多的错误尝试次数, 或者更长的时间间隔, 会导致容错处理开始的时间变的更久, 但是, 并不会影响到错误处理的时间, 无论参数如何变化, 系统重启节点的耗时是一定的.

在CLAIMS系统中, 节点恢复时, 主节点调用从节点脚本, 重启从节点, 从节点重新向主节点发送注册信息, 直到注册成功后, 加入到存活列表.

4 结 论

随着数据分析进“海量数据时代”, 面对越来越大数量级的数据, 分布式系统必将成为未来数据分析的主流载体. 在数据实时分析领域, 目前大部分分布式数据库都不能达到其性能上的要求. CLAIMS 系统作为主要面向金融领域的数据实时分析的内存型数据库, 其可靠性与可用性必然有着更高的要求. 本文及本文所描述的实验, 为 CLAIMS 系统加入了容错机制, 提高了系统的可用性与可靠性, 为其商业应用提供了技术支持. CLAIMS 的容错机制主要由以下几个部分组成:

Fail-fast 阶段: 在 CLAIMS 中, 使用 CAF 来完成节点间的通信, 实现 Fail-fast 机制. CLAIMS 中对于节点的容错需要采用短租约也就是心跳主动去监测节点宕机或网络失效所造的节点失效.

Fail-over 阶段: 在 CLAIMS 中 Fail-over 将保证在节点出现故障时, 不中断地对外服务. 主节点保持监听从节点的活跃状态, 当某个节点不可用(丢失或死亡)时, 主节点将其标记为死亡节点, 同时会终止该从节点上所有未执行的 query 的租约的续租, 所有的从节点都会放弃相关任务, 并接受由调度器新派发的任务继续工作.

Fail-back 阶段: 当一台机器宕机后, 将使其恢复到正常的状态, 继续工作. 目的是恢复原来该机器内存中的数据, 以及已持久化的数据. 当机器未宕机, 只是从节点上的程序崩溃时, 在主节点(Master)Fail-fast 机制中会发现节点失效, 发送命令, 启动存放在指定目录下的重启脚本, 重启该程序. 当出现机器宕机或停电等情况时, 集群管理员重启机器.

[参 考 文 献]

[1] TANENBAUM A S, STEEN M V. Distributed systems principles and paradigms[J]. Acm, 2002, 87(3): 65-73.
[2] COULOURIS G, DOLLIMORE J, KINDBERG T, et al. Distributed Systems: Concepts and Design. [M]. 5th ed. New Jersey: Addison-Wesley, 2012: 37-76.
[3] 王立. 分布式内存数据库系统的查询处理与优化[D]. 上海: 华东师范大学, 2015.
[4] GRAY C, CHERITON D. Leases: An efficient fault-tolerant mechaism for distributed file cache consistency[J]. Acm Sigops Operating Systems Review, 1989, 23(5): 202-210.
[5] CHAROUSSET D, HIESGEN R, SCHMIDT T C. CAF-the C++ actor framework for scalable and resource-efficient applications[C]. New York: ACM, 2014: 15-28.
[6] CASTRO M, LISKOV B. Practical byzantine fault tolerance and proactive recovery[J]. Acm Transactions on Computer Systems, 2002, 20(4): 398-461.
[7] BORTHAKUR D. The hadoop distributed file system: Architecture and design[J]. Hadoop Project Website, 2007, 11(11): 1-10.
[8] 关国栋, 滕飞, 杨燕. 基于心跳超时机制的Hadoop实时容错技术[J]. 计算机应用, 2015, 35(10): 2784-2788.

[9] ZAHARIA M, CHOWDHURY M, DAS T, et al. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing[C]//Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation. Berkeley: USENIX Association, 2012: 141-146.
[10] 林春. 分布式内存数据库的恢复[J]. 航空计算技术, 2003, 33(2): 90-92.

(责任编辑: 张　晶)

(上接第 9 页)

[18] CORBETT J C, DEAN J, EPSTEIN M P, et al. Spanner: Google's globally-distributed database[C]. Operating Systems Design and Implementation, 2012: 251-264.
[19] 阳振坤. OceanBase 关系数据库架构[J]. 华东师范大学学报(自然科学版), 2014(5): 141-148.
[20] 李凯, 韩富晟. OceanBase 内存事务引擎[J]. 华东师范大学学报(自然科学版), 2014(5): 149-163.
[21] 杨传辉. OceanBase高可用方案[J]. 华东师范大学学报(自然科学版), 2014(5): 173-179.
[22] 张晨东, 郭进伟, 刘柏众, 等. 基于Raft一致性协议的高可用性实现[J]. 华东师范大学学报(自然科学版), 2015(5): 172-184.
[23] 庞天泽. 可扩展数据管理系统中的高可用性实现[D]. 上海: 华东师范大学计算机科学与软件工程学院. 2016.
[24] LAMPORT L, SHOSTAK R, PEASE M. The byzantine generals problem [J]. Acm Transactions on Programming Languages & Systems, 1995, 4(3): 382-401.
[25] GRAY J, LAMPORT L. Consensus on transaction commit [J]. Acm Transactions on Database Systems, 2010, 31(1): 133-160.

(责任编辑: 李万会)

(上接第 17 页)

[17] PANDIS I, JOHNSON R, HARDAVELLAS N, et al. Data-oriented transaction execution[J]. PVLDB, 2010, 3(1): 928-939.
[18] In-Memory Operational Database, SQL and Scale-Out. VoltDB[EB/OL]. [2016-06-03]. http://voltdb.com.
[19] Introduction to MemSQL. MemSQL[EB/OL]. [2016-06-03]. http://www.dbms2.com/2012/06/18/introduction-to-memsql/
[20] STONEBRAKER M, MADDEN S, ABADI D J, et al. The end of an architectural era it's time for a completerewrite[J]. VLDB, 2007: 1150-1160.
[21] HELLERSTEIN J M, STONEBRAKER M, HAMILTON J. Architecture of a database system[J]. Foundations and Trends Databases, 2007, 1(2): 141-259.
[22] KEMPER A, NEUMANN T. HyPer: A hybrid OLTP&OLAP main memory database system based on virtual memory snapshots[J]. ICDE, 2011: 195-206.

(责任编辑: 张　晶)