物联网大数据场景下的分布式哈希表适用条件分析

安彦哲[1] 朱妤晴[2),3)] 王建民[1),2),3)]

[1)](清华大学软件学院 北京 100084)

[2)](北京信息科学与技术国家研究中心(清华大学) 北京 100084)

[3)](大数据系统软件国家工程实验室 北京 100084)

摘 要 针对“新基建”带来的物联网大数据管理真实应用场景中的挑战,本文对当前最优实践所用的大规模数据管理系统的核心——分布式哈希表(Distributed Hash Table,DHT),第一次基于极高写入负载和数据流量两个要素,进行了适用条件的理论推导分析.面向存储空间、带宽和时间三方面的限制关系,从理论上分析了写入负载和联网带宽对 DHT 负载再均衡条件的影响,并推导出 DHT 负载再均衡设计仅适用于一定规模的物联网数据管理场景,而不适用于大规模物联网数据管理的结论.利用了基于 DHT 的业界常用系统 Cassandra 的物联网数据负载实验以及系统级模拟器的大量仿真实验结果验证了理论推导结果的有效性.基于理论结果对真实案例进行了应用分析,表明本文的理论结果可用于分析解决当前基于 DHT 系统支撑物联网数据负载出现的问题,并可用于分析和指导物联网数据管理系统的设计.

关键词 物联网数据管理;分布式哈希表;负载均衡;时序数据;时序数据库

中图法分类号 TP311 **DOI 号** 10.11897/SP.J.1016.2021.01679

On Distributed Hash Table's Applicability to Internet-of-Things Big Data Management

AN Yan-Zhe[1)] ZHU Yu-Qing[2),3)] WANG Jian-Min[1),2),3)]

[1)](*School of Software, Tsinghua University, Beijing* 100084)

[2)](*Beijing National Research Center of Information Science and Technology (Tsinghua University), Beijing* 100084)

[3)](*National Engineering Laboratory of Big Data System Software, Beijing* 100084)

Abstract Targeting at the emerging application scenarios and the corresponding challenges of Internet of Things (IoT), this work presents a theoretical analysis on the load rebalancing conditions of distributed hash table (DHT), focusing on the unprecedentedly high workload of writes and the network bandwidth between nodes. While DHT is the state-of-the-practice system structure for large-scale data management, its design has not taken into account the workload characteristics of IoT applications. The typical workload characteristic is the unprecedented intensity of writes. With respect to write workloads and network bandwidth, this paper deduces the applicability conditions of DHT, considering the constraints on bandwidth, storage and time. For DHT-based IoT data management systems with load balancing, the theoretical results imply the following facts: (1) the maximum write throughput that a scalable IoT data management system can support is decided by the number n of nodes to scale to and by the network bandwidth of system nodes; (2) while increasing the number N of system nodes can increase the total storage capacity of the system, it cannot increase the maximum write throughput that the system can support;

收稿日期:2020-10-16;在线发布日期:2021-04-05.本课题得到国家自然科学基金重大项目(71690231)资助.**安彦哲**,博士研究生,主要研究方向为时序数据管理.E-mail:ayz19@mails.tsinghua.edu.cn.**朱妤晴**(通信作者),博士,助理研究员,中国计算机学会(CCF)专业会员,主要研究方向为大数据管理和分布式系统.E-mail:zhuyuqing@tsinghua.edu.cn.**王建民**,博士,教授,博士生导师,中国计算机学会(CCF)高级会员,国家“万人计划”科技创新领军人才入选者,国家杰出青年科学基金入选者,主要研究领域为非结构化数据管理、业务过程与产品生命周期管理、数字版权与系统安全技术、数据库测试技术等.

(3) scale-out processes with a large number n of nodes can lead to sudden and heavy decreases of the maximum write throughput at each system node, leading to disruptive workload redistribution; and, (4) scaling out by a small number n of nodes is a more economical process and complies with the Pay-as-You-Go design consideration of cloud, but still not addressing the problem of scalable IoT data management. Experiments on the widely-used DHT-based system Cassandra and extensive simulations based on standard network system simulator ns-3 validate the theoretical results. With real IoT data management use cases, it is demonstrated that the theoretical results of this work can be used to account for the problems met when exploiting DHT-based systems for IoT data storage, as well as guiding the design of IoT data management system. The results of this paper are applied to analyze the designs of the top 10 time series databases ranked by the DB-Engines website. Among the time series databases that have a distributed version, some have adopted the DHT architecture, e. g. , KairosDB and OpenTSDB; thus, they are expected to come into the problems as described in the paper. The others have circumvented the problem by using other architectures that are not as highly scalable as DHT. A further study of Google's time series database Monarch and IBM's DB2 Event Store shows that, they have abandoned the DHT architecture and chosen layering architectures to avoid the problems under IoT data management workloads. According to the results of this paper, DHT with a load rebalancing design is only applicable to limited-scale IoT data management, but not large-scale IoT data management, especially when the write workload keeps increasing. Unfortunately, as the number of IoT devices keeps increasing, the write workload will inevitably increase. Therefore, a redesign of the DHT load balancing technique or a reconsideration of data distribution architecture is necessary for IoT data management. The results of this paper can be applied to designs, implementations and analyses of large-scale IoT data management systems.

Keywords Internet of Things data management; distributed hash table; load balance; time series; time series database

1 引 言

2020 年以来,国家陆续出台了新型基础设施建设(即"新基建")相关政策,被视为经济转型升级的核心推进器,有望带来新一轮信息化基础设施建设的大浪潮.物联网数据管理与"新基建"的七个核心领域,即 5G 基建、特高压、城际高速铁路和城际轨道交通、充电桩、大数据中心、人工智能、工业互联网等,都具有强相关关系①.物联网数据管理系统是"新基建"的基础软件设施,对于支撑相关领域发展有着至关重要的作用.随着"新基建"的推进,物联网的应用范围将不断扩大,联网的传感器将越来越多,物联网数据管理需求也将愈加急迫.

物联网大数据对其管理系统有以下四点需求:(1)高可扩展性,即系统能在不重新设计架构的前提下扩展至几千个节点,且读写能力随节点扩展呈线性增长;(2)高可用性,即要求系统和数据时时可用,由于数据来自传感器,难以中止等待系统恢复,系统不可用将意味着数据资产的流失;(3)支持海量数据,物联网数据的数据单元为时间序列,一般与传感器有一一映射关系,随着物联网上传感器数量的增加,时间序列的规模通常在数十亿以上,同时,时序数据本身具有高频、流量平稳、超高通量等典型特点;(4)支持弹性扩展和收缩,即系统可在线增加和移除节点,以便于匹配传感器的增加、移除及过期数据的删除等事件.物联网数据管理系统主要为时序数据库管理系统,简称时序数据库.近年来,为了满足物联网数据管理的上述要求和特点,不少时序数据库采用分布式哈希表技术进行设计实现,如 KairosDB②、OpenTSDB③ 等.

分布式哈希表(Distributed Hash Table,DHT)

① "新基建"重塑物联网产业新秩序. http://www.cac.gov.cn/2020-04/09/c_1587975174749164.htm, 2020
② KairosDB. https://kairosdb.github.io/, 2020
③ OpenTSDB. http://opentsdb.net/, 2020

是具备高可扩展性、高可用性、支持大规模存储和弹性伸缩等特点的常用系统架构. DHT 作为基础设施被广泛应用于大规模数据管理系统. 它是分布式键值数据库(Key-Value Store)的基础，比如亚马逊(Amazon)的 Dynamo[1] 和奈飞(Netflix)使用的阿帕奇(Apache) Cassandra[2]，都是基于分布式哈希表 Chord[3] 构建的.

负载均衡是提升系统性能、确保系统可用性的重要机制之一. 对于高可扩展的 DHT 系统，在新增节点后，为确保原有分布式哈希规则不被破坏，系统需要进行负载再均衡. 在分布式键值数据库被广泛应用之前，关于 DHT 负载再均衡的理论分析[3-4]，主要是对主键和节点个数所产生影响进行的讨论，而不涉及与主键关联的数值. 然而，当 DHT 被应用到分布式键值数据库中时，我们不仅需要考虑主键分布和节点个数的影响，也需要考虑数值所代表的大数据产生的影响. 像 Dynamo、Cassandra 这样的键值数据库，一般情况下，其数值所占空间较小，且写入负载相对较轻，因此，不会对系统产生太大影响. 比如，苹果公司所部署的最大 Cassandra 集群有 75000 个节点①，但整个集群的吞吐率只有百万读写每秒. 然而，在物联网条件下，集群仅写入就可以达到近千万每秒的速率[5].

随着 IoT 新场景下写入负载的极大提升，DHT 支持的大规模系统面临着一些新问题. 以下是实践中的一个真实案例：

中车四方案例[地铁智能运维系统场景]——上海地铁有将近 20 条线路，平均每条线路由 10 辆车进行服务. 每辆车装载了约 3200 个传感器来进行状态监测. 每个传感器以 2 Hz 的速率发送数据，每次发送至少 16 字节的数据. 为了将这些数据存储下来进行分析，中车四方所开发的地铁智能运维系统利用了基于 Cassandra 的 KairosDB. 开始时，Cassandra 存储集群有 14 个节点，每个节点每秒需进行约 9 万次写入. 随着地铁监控系统的升级，数据发送频率从 2 Hz 提高到 5 Hz，同时，线路平均车辆数也从 10 辆增加到 15 辆. 因此，存储系统现在需要支撑 480 万次每秒的写入负载. 由于系统容量无法满足当前需求，管理员决定进行扩展，增加一个节点，并在系统扩展的同时允许应用继续写入. 但是，当系统从 14 个节点升级到 15 个节点时，发生了严重的系统问题，而且系统无法再恢复进行服务了.

对于上述案例，在经过系列问题排查后，我们发现并不存在系统实现或使用上的问题. 实际上，系统发生问题不再进行服务的时间段与系统进行扩展、负载再均衡和数据迁移的过程是一致的. 我们在探究问题发生的根源时发现，对于案例中的物联网数据管理应用场景，基于 DHT 的 Cassandra 系统存在设计与应用条件不适配的问题.

分布式哈希表是否适用于物联网数据管理场景，是否能满足物联网数据管理的需求，是物联网数据管理相关研究中必须回答的一个根本性问题. 从该问题出发，本文对基于分布式哈希表的物联网数据管理进行了适用条件分析，主要围绕其负载再均衡过程进行理论推导，并通过仿真模拟实验和真实场景实验对理论推导结果进行了验证，然后基于理论推导结果就应用案例进行了分析，同时对当前常用的分布式时序数据库设计进行了讨论. 基于本文的理论和实验分析结果，对于支持负载均衡的 DHT 系统，存在以下相关结论：

(1) 系统可支持的最大写入吞吐率由扩展节点个数和网络带宽决定.

(2) 系统总节点个数 N 的增加可以导致系统总存储容量的增加，但并不能增加系统可支持的最大写入吞吐率.

(3) 基于较多节点个数 n 的逐次扩展过程将导致单个系统节点的最大写入吞吐率突然降低，使得系统产生剧烈负载波动，影响用户体验.

(4) 基于较少节点个数 n 的逐次扩展过程更经济，也更符合用户期望的“按程收费”(Pay-As-You-Go)理念，但依然不能解决物联网数据管理的根本问题.

本文主要贡献如下：

(1) 面向新的应用场景——物联网数据管理场景，对经典 DHT 系统的负载再均衡过程，在考虑写入负载影响的情况下，重新进行了理论分析. 就作者所知，这是第一次面向物联网场景高写入负载条件下针对 DHT 的适用性理论分析.

(2) 本文基于存储空间限制、带宽限制和时间限制，面向有写和无写负载再均衡过程，推导出基于 DHT 的负载均衡系统可承受的写入负载的三个上限. 这些理论上限表明，支持负载均衡的 DHT 系统并不适用于写入负载会不断增加的物联网数据管理场景.

① Apache Cassandra: Four interesting facts—apple has the biggest cassandra instance. https://www.datastax.com/blog/2019/03/apache-cassandratm-four-interesting-facts, 2020

(3)本文利用经典网络仿真模拟器,模拟了不同高写入负载场景、不同网络系统规模、不同扩展节点规模下 DHT 在存储空间、带宽和时间限制下的运行结果;并利用业界常用的基于 DHT 的 Cassandra,进行了真实场景下的物联网数据负载实验.模拟仿真实验和真实场景实验的结果均匹配了理论结果,从而进一步验证了理论上限的有效性.

(4)基于上述 DHT 系统负载再均衡过程受限的相关理论结果,本文对物联网数据管理实际应用案例进行了原理分析与解释,并就当前常用的分布式时序数据库设计进行了讨论,给出了可有效利用本文理论结果的一般示例,并解释了最新相关研究工作未使用 DHT 架构、而采用多组件的层次架构的根本原因.

本文第 2 节概括相关研究现状;第 3 节描述参考系统模型,并对本文的主要研究问题进行构建;第 4 节面向 n 节点扩展过程,对 DHT 在不同写入负载和带宽条件下的适用条件进行理论推导,并对理论推导结果就应用的两个关注要点进行详细讨论;第 5 节给出模拟仿真实验和真实场景实验的描述及结果分析;第 6 节将本文理论结果应用于实际案例分析和系统设计分析;第 7 节对全文进行总结,并对物联网数据管理系统设计相关研究作出展望.

2 研究现状

物联网大数据的主体是时序数据(又称时间序列),由物联网上的每个传感器定时发送的时间与数值对序列构成.时序数据频率极高,数量极大,且存在特定数值操作,因此,需要专门的时序数据库①进行管理,常见的关系数据库、键值数据库等都不适合.为了满足物联网数据的管理需求,时序数据库还必须具备分布式与可扩展能力.

相关时序数据库.开源的分布式时序数据库一般基于现有高可扩展系统构建,如 KairosDB 基于 Cassandra[2]、OpenTSDB 基于 HBase②、TimescaleDB③基于分布式 PostgreSQL④ 等.而现有高可扩展系统一般利用分布式哈希表的特点进行设计和实现.闭源的分布式时序数据库中,近期公开发表的有谷歌(Google)的 Monarch[6] 和 IBM 的 DB2 Event Store[7].其中,谷歌的 Monarch 以其之前的 Slicer[8] 系统为基础,明确拒绝了基于 DHT 的负载均衡模式,并给出了经验性的理由,但没有从理论上给出根本性的原因;而 DB2 Event Store 将存储可扩展的问题留给了底层的共享存储来解决.另外,业界使用较多的 InfluxDB⑤ 虽然也基于特定策略进行数据分布,但在具体实现中并不进行负载再均衡,与 DHT 的模式不尽相同.

分布式哈希表(DHT).Chord[3]、CAN[9] 和 Pastry[10] 是最知名的分布式哈希表系统,它们有不同的主键到节点的映射方式,但是,其基本核心都是一致性哈希.DHT 在联网的分布式节点集群上,提供了一层类似于哈希表的服务,使得访问集群中的任意节点都可以通过给定主键确定数据所在节点.若允许数值随主键同样存储在对应的节点上,则基于一致性哈希的特性[11],当主键个数达到一定数量时,数据将在集群节点上均匀分布,从而使系统获得对应的负载均衡特性.由于允许通过任意系统节点进行数据访问,因此 DHT 不存在单点瓶颈;同时,系统可通过增加节点进行扩展,数据将依据负载均衡策略进行移动,使得系统原有的分布式哈希规则得到有效保证.因此,基于 DHT 的系统都具有高可扩展性.

在实际中,DHT 被应用于很多主流系统,如键值系统[12] 和文件系统[13].亚马逊的 Dynamo[1] 和阿帕奇的 Cassandra[2] 是最广为人知的基于 DHT 的系统.由于具有高可扩展性和负载均衡特点,因此它们被部署在很多云端系统中.这些基于 DHT 的系统通过调整主键-节点的映射方式来获得更好的负载均衡特性.在本文中,我们将主要讨论采用了负载均衡设计的 DHT 系统.

DHT 负载均衡机制.早年间,对 DHT 的理论分析大都只涉及负载均衡[14]、路由代价[15]、成员变更处理[16] 与恶意攻击[17] 等方面.虽然负载再均衡的问题也曾得到讨论[18],但是,这些工作都是面向成员变更和主键搜索负载开展的,写入负载对于系统的影响未曾被引入到相关理论分析中.类似地,尽管存在对分布式键值数据库的负载均衡分析工作[19],然而写入负载的影响始终未得到相关理论的考虑.究其原因,早期 DHT 主要是用于做分布式对象的索引,而键值数据库未曾遇到过像 IoT 场景那样的高强度写入负载——过去的假设是键值数据库负载以读为主[20].

① Db-engines ranking of time series dbms. https://db-engines.com/en/ranking/time+series+dbms, 2020
② HBase. https://hbase.apache.org/, 2020
③ TimescaleDB. https://blog.timescale.com/blog/building-a-distributed-time-series-database-on-postgresql/, 2020
④ PostgreSQL. https://wiki.postgresql.org/wiki/Replication, Clustering, and Connection Pooling, 2020
⑤ InfluxDB. https://www.influxdata.com/, 2020

基于常见的可扩展系统基础设施 DHT，本文从理论的角度对分布式时序数据库的设计与应用条件进行分析与研究. 与以往工作不同，本文对 DHT 的理论推导与分析引入了新的应用场景，即物联网数据管理场景，分析了 DHT 负载再均衡特点在高写入压力条件下所受到的影响，从而得出了与过去工作不同的结论. 本文的分析不对物联网的具体应用条件作假设，因而适用于广泛的物联网场景，相关理论结果可被用于指导物联网数据管理系统设计与应用.

3　系统模型与问题定义

3.1　准备知识

分布式哈希表通过一致性哈希[11]将 K 个主键均匀分布在 N 个网络相连的节点上. 每个节点分配 T 个令牌. 这些令牌按其值在一维数值空间的位置被映射到在空间中划分的不同主键范围. 每个节点将存储一维数值空间中其令牌与相邻下一令牌所划定范围之间的主键. 同时，每个节点维护 $O(\log N)$ 个其它节点的路由信息. 在进行主键查询时，用户最多只需要和 $O(\log N)$ 个节点通信，即可定位主键所在节点. 由于每个节点只需维护有限个节点的路由信息，因此，当系统成员发生变动时，仅需要非常有限的工作即可实现系统的重新调整. 这一特点，使得基于 DHT 的系统可以高效地扩展到上千的节点规模.

为了满足云计算应用的需求，云端键值数据库对 DHT 的设计进行了一些改动. 这些改动主要是为了获得更好的负载均衡特性，以满足云计算应用的基本要求. 首先，一个节点不仅维护它负责的主键相关的路由信息，同时也存储这些主键所对应的值，这使得当主键满足均匀分布时，数据也可以均匀地分布在节点上. 其次，当系统发生成员变化时，数据也随着主键一起移动到新的负责节点上，这使得系统可以在成员发生变更时继续保持负载均衡特性. 最后，主键与节点的映射也随着负载再均衡过程进行相应调整.

负载再均衡是系统节点成员发生变化时进行的数据移动过程. 当系统需要进行容量扩展时，系统将增加节点成员. 本文主要关注系统在扩展时发生的负载再均衡过程. 但是，因为系统收缩过程是扩展过程的逆过程，所以本文的结果简单调整后，也同样适用于系统收缩过程. 当扩展事件触发时，系统将处于一个不稳定的过程——数据在节点间移动，节点调整各自的路由信息. 我们也把这一过程称为稳定化过程，因为系统在这一过程通过稳定化来维持系统的特性. 系统稳定化过程在系统成员发生变更和数据迁移时开始. 当路由信息和节点上的数据状态与系统设计所定义的条件相匹配时，系统稳定化过程结束. 根据系统设计所定规则，系统中的负载应当可以均匀分布.

负载分布有三种常见的策略[1]. 它们的主要差别在于如何将令牌赋予节点以及如何进行主键范围分割. 根据这两方面的不同选择，我们将这三种策略命名为有限令牌随机分割、有限令牌均匀分割和多令牌均匀分割. 第一种“有限令牌随机分割”的策略为每个节点分配 T 个随机令牌，将按照令牌在主键空间的位置分割主键范围. 第二种“有限令牌均匀分割”的策略为每个节点分配 T 个随机令牌，并将主键空间均匀分割成许多主键范围，再按照令牌在主键空间的位置，将每个主键范围分配给 r 个对应的令牌. 第三种“多令牌均匀分割”将主键空间分割成 Q 个等大的范围，并为每个节点随机分配 $\frac{Q}{N}$ 个令牌. 第一种策略是 DHT 原始的主键-令牌映射策略，因此它可以支持高可扩展性. 但是，第三种策略在负载均衡及元数据大小方面是表现最好的，因此，它被云端系统广泛采用[1-2].

对于“多令牌均匀分割”策略，主键空间被分成 Q 个相等主键范围. 每个范围分割的最右边界被作为令牌，因此有 Q 个令牌. 每个节点被随机指定给 $\frac{Q}{N}$ 个令牌. 因此 Q 不一定需要很大，而第二种策略则要求 Q 必须足够大；这也使得第三种策略需要维护的元数据信息比第二种少. 当指定副本数为 r 时，一个分割范围将被映射到 r 个不同的节点上，这 r 个节点是从这一分割范围开始，绕着主键空间构成的环沿顺时针方向碰到的前 r 个不同节点. 当一个节点加入系统时，这个节点从另外 N 个节点中的每一个的令牌集中随机抽取 1 个来形成它的令牌集. 当节点离开系统时，离开的节点将它的令牌集随机分配给剩余的其它节点. 本文以下讨论均假设基于这一策略.

3.2　系统模型

一个基于 DHT、包含 N 个节点的系统，最初处于稳态，存储 K 个主键及其对应的数值. 当系统扩展加入节点后，系统需要继续维持稳态. 从一个稳态过渡到另一个稳态的过程称为稳定化过程. 根据相

关研究[11]，基于一致性哈希构建的系统，在处于稳态时，满足以下定理.

定理 1. K 个主键哈希到 N 个节点，则以下事件大概率为真：

(1) 每个节点最多负责$(1+\epsilon)\frac{K}{N}$个主键.

(2) 当第$(N+1)$个节点加入或离开系统时，有$O\left(\frac{K}{N}\right)$个主键的负责节点发生了变化，且仅是主键迁移到加入节点或主键迁移出离开节点.

一致性哈希的相关论文还证明了，通过让每个节点模拟 $\Omega(\log N)$个虚拟节点，可以让ϵ任意小. 在实际中，虚拟节点可以通过令牌来表示. 因此，如果每个物理节点负责多个令牌，则 K 个主键在系统中可以几乎均匀地分布到 N 个节点上. 基于上述定理，我们还可以得到以下推论.

推论 1. K 个主键哈希到 N 个节点，则以下事件大概率为真：当第$(N+1)$到第$(N+n)$个节点同时加入或同时离开系统，且 $N\gg n$ 时，有 $O\left(\frac{K}{N}\right)$个主键的负责节点发生了变化，且仅是主键迁移到加入节点或主键迁移出离开节点.

基于上述定理和推论，当采用“多令牌均匀分割”负载分布策略时，每个节点将负责$\frac{K}{N}$个主键及这些主键对应的数值. 假设键值大小的均值为 v，则每个节点存储的数据大小为 $v\frac{K}{N}$. 假设存在 r 个副本，则每个节点将同时负责 $r\frac{K}{N}$个主键，并需要提供大小至少为 $rv\frac{K}{N}$的存储空间.

系统会在特定条件下启动一个扩展过程，如写入负载太重，以至于当前节点无法及时进行处理. 每个系统扩展过程都涉及 n 个节点的加入过程. 假设系统启动时刻为 0，在扩展之前的写入负载为每节点单位时间内平均处理 λ 写入，则在系统扩展时刻 T_{N+n}，系统有 K 个键值对，且以下关系成立：

$$K=\lambda N T_{N+n} \tag{1}$$

3.3 问题定义

本文从中车四方案例出发，主要针对基于 DHT 的系统在扩展过程中发生失效的问题，探讨 DHT 系统在物联网数据管理场景的适用条件. 面向系统扩展所涉及的负载再均衡过程，我们对该问题进行梳理和分析. 因为扩展过程中系统的写入负载未曾增加，所以，系统宕机的原因可能是过载，而过载则是由于负载再均衡加上写入负载的合并压力产生的. 为此，我们考虑两种情况，一是系统在负载再均衡过程中不接收写，但写入负载依然存在，只是需要系统在稳定化之后一并处理. 另一种情况则是系统在负载再均衡过程中接收写入. 我们假设这两种情况下系统稳定化前后的写入负载不发生变化.

接下来，我们分别针对系统受到的三方面限制进行分类讨论，即存储空间限制、带宽限制和时间限制. 在讨论过程中，我们将对以下问题进行分析：

(1) 在负载再均衡过程中，哪些要素对基于 DHT 的系统产生了最大的影响？

(2) 这些要素存在什么样的限制？

本文的理论分析主要讨论写入负载，暂不考虑查询负载对这一过程的影响. 但是，因为我们是从存储空间限制、带宽限制和时间限制这三方面开展的分析，所以即使增加了查询负载，分析结果也是适用的. 只是查询负载的增加会导致更紧致的分析上限.

本文所涉及的主要符号总结如表 1. 在不同规模的大数据系统中，这些符号所代表的变量的取值不同，但本文分析过程及结论不依赖于它们的具体取值，具有一般适用性.

表 1 文中主要符号与定义

符号	定义
N	原有节点个数
n	新增节点个数
λ	单个节点的写入速率
K	主键个数
S	单个节点的存储空间
r	副本个数
v	键值对大小
μ	达到负载再均衡条件时，单个节点已使用存储空间占其总存储空间的比例
b	单个节点的带宽

4 物联网场景下大数据系统扩展过程中的 DHT 适用条件推导

为了使讨论具有一般性，本文面向系统从 N 个节点扩展到 $N+n$ 个节点的情况进行讨论，其中，$N\gg n$，且讨论主要围绕问题核心——负载再均衡过程展开. 首先对系统扩展过程中成立的一般关系进行讨论.

主键迁移. 系统稳定化过程在系统扩展时发生，主要是为了保证负载再均衡. 在系统稳定化过程中，数据会从原来所在的 N 个节点迁移到新加入的 n 个节点. 基于“多令牌均匀分割”策略，新加入的 n

个节点中的每一个都将从 N 个节点收到 k_{N+n} 个主键. 基于上述系统模型，可以得到以下公式：

$$k_{N+n}=\frac{rK}{N+n} \tag{2}$$

数据迁移. 每个新加入的节点从 N 个节点处收到的总数据量为 d_{N+n}. 本文假设扩展过程都以键值为单位移动，实际上系统是以“键值范围”(Q 相关)来移动数据的. 但是，当 Q 和 K 都足够大时，即数据量很大时，两者的差别可以忽略. 因此，基于式(2)，我们可推知以下关系：

$$d_{N+n}=\frac{vrK}{N+n} \tag{3}$$

存储空间. 假设每个节点可用存储空间大小为 S. 当第 $N+1,\cdots,N+n$ 个节点加入时，系统原来的每个节点平均各使用了 μS 的存储空间. 因此，d_{N+n} 与存储空间之间的关系，还可以表达为下述方程：

$$d_{N+n}=\frac{\mu NS}{N+n} \tag{4}$$

基于式(3)和(4)，我们可以推出以下关于系统存储空间及主键个数的方程：

$$\mu NS=vrK \tag{5}$$

$$K=\frac{\mu NS}{vr} \tag{6}$$

4.1　有写扩展过程

假定系统在扩展过程中，写入以 λN 的均匀速率到达，并均匀分布到各个系统节点进行处理，则以下两个定理成立.

定理 2.　分布式哈希表从 N 个节点扩展到 $N+n$ 个节点，扩展过程中系统总写负载保持不变，为 λN，每个节点带宽为 b，写入负载的数据单元大小为 v，扩展前每个节点已使用存储空间占其总存储空间比例为 μ，为避免扩展过程中所有节点的剩余空间被占满而引发异常，则系统扩展前每个节点单位时间内可接受写入频率受以下不等关系约束.

$$\lambda<\left(1-\mu+\frac{n}{N}\right)\frac{b}{v} \tag{7}$$

定理 3.　分布式哈希表从 N 个节点扩展到 $N+n$ 个节点，扩展过程中系统总写负载保持不变，为 λN，每个节点带宽为 b，写入负载的数据单元大小为 v，假设系统扩展的触发条件为平均每个节点已使用空间占其总存储空间比例达到 μ，为避免扩展过程的交叠而引发异常，则系统扩展前每个节点单位时间内可接受写入频率受以下不等关系约束.

$$\lambda<\frac{n}{N}\frac{b}{v} \tag{8}$$

下面，给出上述两个定理的推导证明思路. 首先对两个定理推导的公共前提进行定义.

假定系统稳定化过程持续时间为 t_{N+n}. 由于负载是均衡的，因此每个新加入节点将需要处理的写入为 $\frac{\lambda N}{N+n}t_{N+n}$. 结合式(2)，在系统稳定化过程中，到达每个新加入节点的总数据量为

$$D_{N+n}=\left(\frac{rK}{N+n}+\frac{\lambda N}{N+n}t_{N+n}\right)v \tag{9}$$

假设每个节点有足够的带宽来传输数据到新加入的节点. 令 b 为每个节点的总带宽，b_{N+n} 为每个新加入节点用于处理稳定化过程所涉及数据传输的带宽，剩余的带宽都用于处理新接收的写入. 基于式(3)，我们可以得到以下等式：

$$b-b_{N+n}=\frac{\lambda N}{N+n}v \tag{10}$$

$$t_{N+n}=\frac{vrK}{(N+n)b_{N+n}} \tag{11}$$

基于式(9)和(11)，我们可算得迁移到每个新加入节点的总数据量为

$$D_{N+n}=\frac{vrK}{N+n}\left(\frac{\lambda vN}{(N+n)b_{N+n}}+1\right) \tag{12}$$

4.1.1　基于存储空间限制的分析

对于一般系统而言，我们都有以下事实上的存储空间限制，即：

命题 1.　存储空间限制. 在系统稳定化过程中，应用写入负载不能把所有节点的剩余空间占满.

上述命题所表达的意思是，写入占满所有节点剩余的空间所耗费的时间应当大于 t_{N+n}. 下面我们将利用上述命题推导 λ 与 b 之间的关系. 命题 1 可以表达为以下公式：

$$\frac{(N+n)S-\mu NS}{\lambda vN}>t_{N+n} \tag{13}$$

将式(6)和(11)和代入式(13)，我们可以得到以下结果：

$$\left(\mu^{-1}-1+\frac{n}{\mu N}\right)b_{N+n}>\frac{\lambda vN}{N+n} \tag{14}$$

将式(10)代入式(14)，我们会得到：

$$b_{N+n}>\frac{\mu N}{N+n}b \tag{15}$$

基于式(10)和(15)，我们可以推导得到式(7)，从而定理 2 得证.

根据定理 2，系统可以处理的写入负载速率 λ 与系统大小，即节点个数是相关的. 同时，还与系统扩展时每个节点平均已占用空间相关. 从式(7)，我们也可以得知，增加同时扩展的节点个数 n，可以提升系统可处理写入负载的速率上限. 但是，n 也是受限制的，首先它要小于 N，其次基于上述分析的前

提假设，我们还必须确保 $N \gg n$，也即 n 不能是一个很大的数值.

4.1.2 基于带宽限制的分析

假定系统触发扩展的条件是，每个系统节点本地所存储数据量达到 μS. 对于一般系统而言，我们知道存在以下带宽限制：

命题 2. 带宽限制. 对于系统扩展前的 N 个成员节点，其数据迁移至新加入节点的速率要高于写入负载数据到达的速率.

若上述带宽限制条件得不到满足，则会发生原成员节点在系统再均衡过程中存储空间被占满的现象. 同时，由于原成员节点存储数据量始终大于 μS，因此系统将一直处于扩展过程中.

在系统扩展之前，每个节点的写入负载为单位时间 λ；在新节点加入之后，每个系统节点（包括新加入节点）所承受的写入负载将降为 $\frac{\lambda N}{N+n}$. 根据式(4)，命题 2 可以表达为以下公式：

$$\left(\mu S-\frac{\mu NS}{N+n}\right)\frac{1}{t_{N+n}}>\frac{\lambda vN}{N+n}\Rightarrow \frac{\mu nS}{(N+n)t_{N+n}}>\frac{\lambda vN}{N+n} \tag{16}$$

基于式(6)、(11)和式(16)，我们可以推导出以下关系：

$$b_{N+n}>\frac{\lambda vN^2}{n(N+n)} \tag{17}$$

将式(10)代入上述不等式，可推得式(8)，即定理 3 得证.

由定理 3，我们可以推知，系统的整体写入负载 λvN 必须严格小于因为新增节点而扩充的带宽 nb. 从另一个角度讲，系统当前的写入速率不能大于新增节点全部带宽用于处理写的情况下能处理的写入速率. 上述条件被违背的结果是，在系统进行负载再均衡时，节点存储空间将在稳定化过程完成之前被占满，或者系统扩展过程再次被触发，从而导致系统永远无法进入稳定化运行状态.

同时，由于 $\lambda<\frac{b}{v}$，再考虑式(8)，即使让 $n>N$，也不能使得系统支持更大的写入负载. 也就是说，如果考虑系统负载再均衡，那么系统每次扩展过程可以增加的节点个数，应当不大于系统扩展前节点个数更合适.

4.2 无写扩展过程

对于无写扩展过程，我们假设系统在此过程中不需要处理写入，但写入仍然以 λN 的均匀速率到达系统，并积累等待扩展过程结束后，系统再进行处理，则以下定理成立.

定理 4. 分布式哈希表从 N 个节点扩展到 $N+n$ 个节点，扩展过程中系统总写负载保持不变，为 λN，每个节点带宽为 b，写入负载的数据单元大小为 v，假设系统扩展的触发条件为平均每个节点已使用空间占其总存储空间的比例达到 μ，扩展过程中节点不处理写入，等待扩展结束后再处理写入，为确定系统可以及时处理完成写入，则系统扩展前每个节点单位时间内可接受写入频率受以下不等关系约束.

$$\lambda<\frac{n}{N}\frac{b}{v} \tag{18}$$

下面，给出上述定理的推导证明思路. 根据无写扩展过程的设定，我们可以得到以下等式关系：

$$D_{N+n}=d_{N+n} \tag{19}$$

$$b=b_{N+n} \tag{20}$$

基于上述两个等式以及式(4)，我们可以推导出系统用于稳定化过程的时间如下，即每个新加入节点都将所有带宽用于负载再均衡：

$$t_{N+n}=\frac{d_{N+n}}{b}=\frac{\mu NS}{(N+n)b} \tag{21}$$

假定系统扩展之前，每个节点的写入负载为单位时间 λ，则在系统稳定化过程结束的时候，写入累积的数据可以表达为

$$d_{\mathrm{acc}}=t_{N+n}\times\lambda vN=\frac{\mu\lambda vN^2}{(N+n)b}S \tag{22}$$

假设在系统稳定化过程之后，每个节点用于追赶累积写入的带宽为 b'_{N+n}. 那么，基于式(22)的假设，可得到以下等式：

$$b'_{N+n}=b-\frac{\lambda vN}{N+n} \tag{23}$$

基于式(22)和(23)，我们可以计算出系统将累积的写入处理完成所需的等待时间：

$$t_{\mathrm{catch\_up}}=\frac{d_{\mathrm{acc}}}{(N+n)b'_{N+n}}=\frac{\mu\lambda vN^2}{(N+n)b}\frac{S}{(N+n)b-\lambda vN} \tag{24}$$

因此，随着系统大小 N 的增加，用于处理累积写入的时间也将增加.

在一个无写稳定化过程结束后，系统存储的数据量将达到 μNS，同时，系统将总共拥有大小为 $(N+n)S$ 的存储空间. 那么，在触发下一次扩展之前，系统还可以再存储大小为 μnS 的数据.

假设系统从上一个稳定化过程结束到下一个稳定化过程开始，中间间隔的时间为 Δt. 在这段时间内，由于既要处理累积写入，又要接受负载为 λvN

的常规写入，那么系统的平均写入负载一定大于 λvN；同时单个节点又会受到最大带宽 b 的限制，因此 Δt 满足以下关系：

$$\Delta t\in\left[\frac{\mu nS}{(N+n)b},\frac{\mu nS}{\lambda vN}\right) \tag{25}$$

4.2.1　基于时间限制的分析

在时间方面，我们知道以下事实必须得到满足：

命题 3.　时间限制. 系统处理完成稳定化过程中的累积写入的时刻应当要早于下一次系统扩展的时刻.

否则，系统的扩展过程将连续被触发，从而导致系统可用于写入的带宽将持续减少，并将在某次扩展之后再也没有剩余带宽用于接收新的写入. 根据时间限制，我们可以得到以下不等关系：

$$\Delta t>t_{\text{catch\_up}} \tag{26}$$

根据不等式(25)可知，不等式(26)可转化为 $\Delta t_{\min}>t_{\text{catch\_up}}$，由此我们可以推导出以下不等式：

$$\frac{\mu nS}{(N+n)b}>\frac{\mu\lambda vN^2}{(N+n)b}\frac{S}{(N+n)b-\lambda vN} \tag{27}$$

将上述不等式整理后可以得到不等式(18)，从而定理 4 得证.

4.3　写入上限讨论

至此，我们已经从理论上分析了物联网场景下大数据系统进行 n 节点扩展时，其对写入负载上限 λ 的要求. 根据定理 2、定理 3 和定理 4，关于系统可承受写入负载 λ 一共有三个上限结论，即式(7)、(8)和式(18). 其中，由定理 3 和定理 4 得到的上限均为 $\frac{n}{N}\frac{b}{v}$，且小于由定理 2 得到的上限 $\left(1-\mu+\frac{n}{N}\right)\frac{b}{v}$.

由此，我们可推知：基于 DHT 的系统若采用了负载再均衡，则其可承受的最大写入上限要同时受到原系统规模 N 及新增系统节点个数 n 的限制.

同时，根据定理 2、定理 3 和定理 4，同样扩展 n 个节点，随着 N 的增大，系统扩展前平均每个节点可接受写入频率 λ 的上限均有所下降. 若触发条件为每个系统节点本地所存储数据量占其总存储空间比例达到 μ，则根据式(4)，系统在进行扩展时，随着 N 的增大，同样新加 n 个节点，则负载再均衡数据量将增加，而新加节点用于接收这部分数据量的带宽也必须增加. 由于单个节点总带宽 b 和键值对大小 v 保持不变，则根据式(10)，只能减少 λ.

4.4　扩展模式讨论

接下来，我们基于带宽限制和时间限制下的 λ 上限 $\frac{n}{N}\frac{b}{v}$，分别对系统进行单次扩展和多次扩展的情况进行讨论. 存储空间限制下的讨论同理.

单次扩展. 基于式(8)和式(18)，可知扩展前大小为 N 的系统与扩展后大小为 $N+n$ 的系统，它们可承受的单个节点写入上限之差 $\Delta\lambda$ 满足：

$$\Delta\lambda=\frac{n}{N}\frac{b}{v}-\frac{n}{N+n}\frac{b}{v}=\frac{n^2}{N(N+n)}\frac{b}{v} \tag{28}$$

根据公式，随着 n 的增大，$\Delta\lambda$ 也会增大. 当 n 为 1 时，$\Delta\lambda$ 取得最小值. 如果 $N\gg n$，那么由于系统扩展引发的单个节点最大写入上限的下降几乎可以忽略不计；但是，如果 n 与 N 很接近，那么系统单个节点最大写入上限将会大幅下降，从而导致系统负载分布在扩展前后发生剧烈变化；更极端的情况是 $n>N$，稳定化过程结束后，单个节点最大写入上限将会减少一半以上，也即应用对系统资源的有效利用率降低到 50%以下，这对于实际应用而言将是难以接受的.

另一方面，当 n 与 N 很接近时，定理 1 的第二个结论将变为：当 n 个节点同时加入或离开大小为 N 的系统时，有 $O\left(\frac{K}{2}\right)$ 个主键的负责节点发生了变化，且仅是主键迁移到加入节点或主键迁移出离开节点. 如果 K 的值很大，那么半数主键迁移将会大量占用系统带宽等资源，从而使得实际应用对系统资源的有效利用率大幅降低. 当 n 与 N 相等时，主键迁移最理想的状态是 N 个节点中的每一个都将其一半数据迁移到另外一个新增的节点. 但由于 DHT 系统本身哈希规则的作用，这一理想状态往往难以实现. 因此，每个新增节点都会随机地和多个原有节点进行通信，消耗的资源和时间也会进一步增加.

连续多次扩展. 将上述的单次扩展依次串接起来，我们可以推得连续多次扩展的结果.

(1) $1\leq n\leq N$. 根据式(8)可以得知，系统整体的写入上限恒为 $\frac{nb}{v}$，与 N 无关，这也就意味着系统可以支撑的写入负载不会因为节点的增加而增加. 如果想要系统始终保持较高的写入速率，那么每次扩展都应增加 $O(N)$ 个节点，这样做会导致累积的节点个数将以指数级的速率增长，有悖于 DHT 系统高可扩展性及弹性的设计初衷，更与用户期望的“按程收费”理念相违背.

(2) $n>N$. 根据 4.1.2 节的结论，无论 n 如何增长，系统都无法支持更大的写入负载. 这种系统扩展规模的情形并不符合实际应用要求和条件.

综上所述，对于支持负载均衡的 DHT 系统，在进行扩展时，选择一个较小的 n 值是更为合理的选择. 因为，虽然本文定理 2、定理 3 和定理 4 给出的

三个 λ 的上限，在 $n \leq N$ 时均成立，但是如果 n 被设置得与 N 很接近或大于等于 N，则会导致系统节点负载发生剧烈突变、增加过多节点间通信、违背系统弹性设计原则等种种问题. 而且，根据本节及 4.3 节的讨论，对于支持负载均衡的 DHT 系统，增加节点可以带来存储容量方面的收益，但并不会导致系统支持更大的写入负载. 因此，支持负载均衡的 DHT 系统并不适用于写入负载会不断增加的物联网数据管理场景.

5 实验验证

本节分为仿真实验验证和真实场景实验验证两部分. 仿真实验针对三种不同的条件限制，面向远高于常见情形的物联网数据应用负载，对多种大规模集群系统部署环境进行模拟仿真. 为了使结果可以直接与实际案例相匹配，仿真实验基于中车四方案例的具体部署环境信息进行复现与验证. 两部分的实验结果均验证了第 4 节中结论的有效性.

5.1 仿真实验验证

5.1.1 仿真验证方法与配置

仿真验证实验基于离散事件网络模拟器 ns-3[①] 开展. 网络模拟器 ns-3 是经典的网络仿真软件，获得了 SIGCOMM 的 2020 年网络系统大奖. 在利用 ns-3 的仿真模拟实验中，主要模拟实际物理节点的存储及其之间的数据传输. 基于中车四方案例的具体信息，考虑节点规模为 15 个节点的情形，包括 14 个原有节点和 1 个新加节点. 对于有写扩展过程，如图 1 所示，全体原有节点均会向新加节点传输数据，同时外部数据源与系统内各个节点之间存在数据传输. 对于无写扩展过程，如图 2 所示，在稳定化过程中，全体原有节点均会向新加节点传输数据；

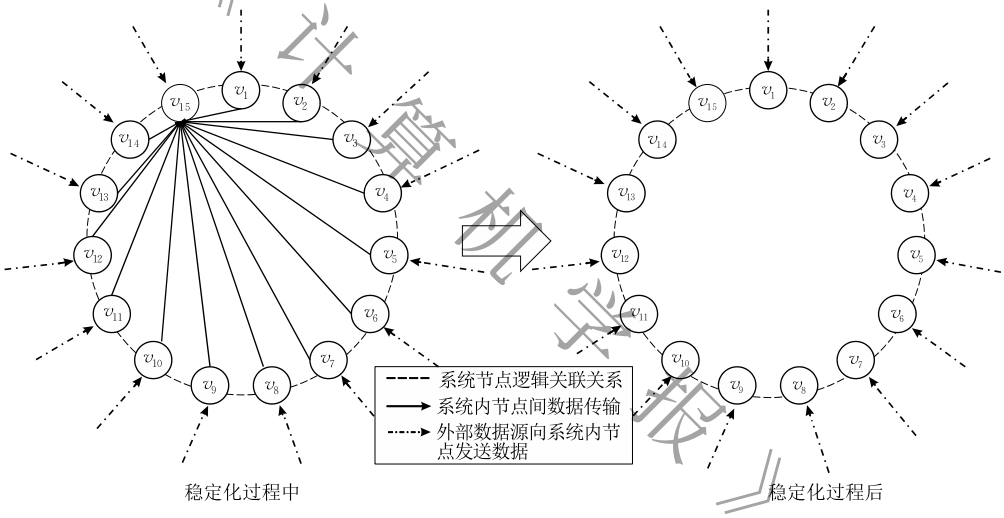

图 1 有写扩展过程的系统拓扑模型变化

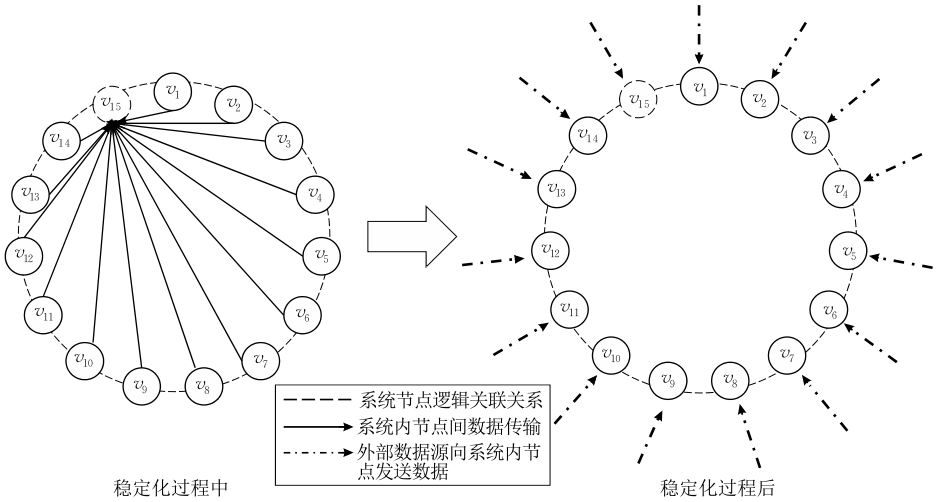

图 2 无写扩展过程的系统拓扑模型变化

① ns-3. https://www.nsnam.org/, 2020

稳定化过程结束后，系统内全体节点需要处理累积的写入数据并接收新的写入.其中，节点之间的有向边代表了数据传输关系，有向边起点为数据发送方，终点为数据接收方.

仿真验证使用了 TCP 协议模拟器.仿真参数设置如下：数据包大小为 1 KB，单个节点容量上限为 10 GB；加入新节点时原有节点平均存储数据量为 8 GB；系统存储数据量达到容量上限的 80%时会触发负载再均衡；单个节点的带宽为 1 Gbps.另外，为了与第 4 节的前提保持一致，我们假设每个原有节点向新加节点传输数据的速率相等且恒定，外部数据源向系统内每一个节点传输数据的速率相等且恒定.

为了覆盖第 4 节讨论的三种限制条件，我们设计了以下三种仿真场景：(1) 面向有写扩展过程，系统扩展过程由用户主动发起；(2) 面向有写扩展过程，系统扩展过程基于存储容量触发；(3) 面向无写扩展过程，系统扩展过程基于存储容量触发.在上述三种场景中，我们分别对多种写入负载以及系统规模做了模拟仿真.在每一种场景中，写入负载分别取大于、接近和小于该限制条件下的上限的三个值；系统规模分别取 21、14 和 7，即大于、等于和小于中车四方案例中的节点个数.另外，我们还在场景(2)中增加了一组实验：在系统规模为 14 时，设置新增节点个数分别为 1、7 和 14，以此来模拟不同新增节点个数对系统稳定化过程的影响.

5.1.2　仿真实验结果

存储空间限制条件.在系统规模为 14 的条件下，我们分别设置实际写入负载为单个节点 500 Mbps、300 Mbps 和 100 Mbps，则剩余的带宽用于负载均衡.对新加节点以及任一原有节点的存储数据量进行监控，结果如图 3 所示，其中×表示真实场景下系统宕机的时刻，即曲线进入灰色区域后系统将失效.A 点、B 点和 C 点分别表示三种情况下系统稳定化过程结束的时刻.根据 4.1.1 节的结论，我们可以得到系统规模为 14 时，单个节点的写入负载上限为 270 Mbps.从图 3 我们可以得知，当实际写入负载小于该上限时，系统会在达到容量阈值之前就结束稳定化过程；反之，系统在稳定化过程中就会把所有节点的剩余空间占满.

在写入负载为 250 Mbps 的条件下，我们分别设置系统节点个数为 7、14 和 21.对新加节点以及任一原有节点的存储数据量进行监控，结果如图 4 所示.经计算，当系统规模为 7 和 21 时，单个节点的写入负载上限分别为 340 Mbps 和 250 Mbps.从图 4 我们可以得知，在相同的写入负载下，系统规模越大，其写入负载上限越低，完成稳定化过程需要的时间越多，从而在稳定化过程结束之前占满所有剩余空间的风险就越大.

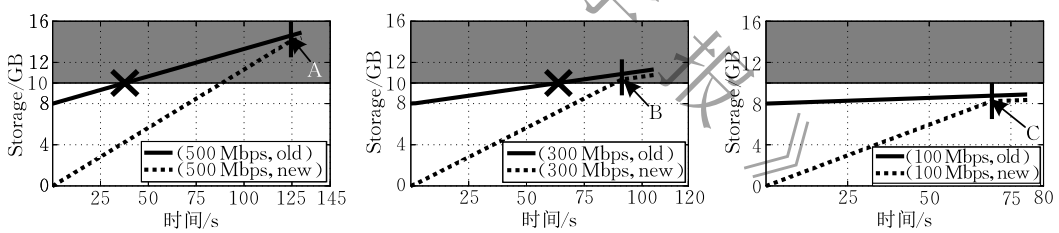

图 3　存储空间限制：不同写入速率 λ(灰色区域为失效区，竖线标志位置为稳定化过程结束的时刻)

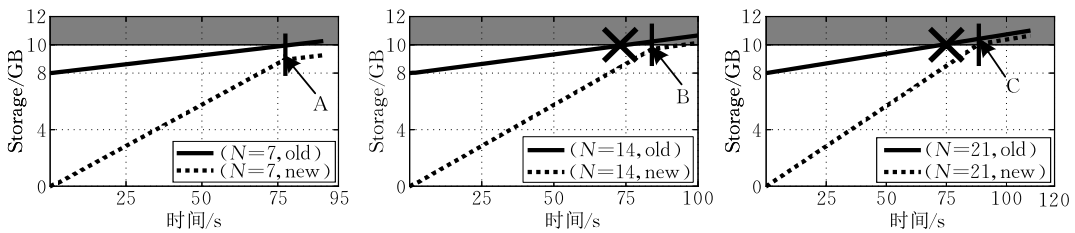

图 4　存储空间限制：不同系统规模 N(灰色区域为失效区，竖线标志位置为稳定化过程结束的时刻)

带宽限制条件.在系统规模为 14 的条件下，我们分别设置单个节点实际写入负载为 100 Mbps、70 Mbps 和 40 Mbps.监控新加节点以及任一原有节点的存储数据量，结果如图 5 所示.根据 4.1.2 节的结论，我们可以得到系统规模为 14 时，单个节点的写入负载上限为 70 Mbps.从图 5 我们可以得知，当实际写入负载大于该上限时，原有节点的存储数据量始终不会低于触发系统扩展的容量阈值，那么系统将会一直处于稳定化过程中；反之，原有节点的存储数据量会先减小，等到稳定化过程结束后再增加.可以看到，尽管节点带宽可达 1 Gbps，但在系统规模为 14 时，其可接受的合法写入速率必须在带宽的

1/14 之下.因此,实际上系统对带宽资源的利用率很低,且该利用率随着系统规模的增加将继续降低.

在写入负载为 100 Mbps 的条件下,我们分别设置系统节点个数为 7、14 和 21.对新加节点以及任一原有节点的存储数据量进行监控,结果如图 6 所示.经计算,当系统规模为 7 和 21 时,单个节点的写入负载上限分别为 140 Mbps 和 50 Mbps.从图 6 我们可以得知,当系统规模为 14 和 21 时,系统实际写入速率大于写入负载上限,所以会出现扩展过程交叠的现象;而系统规模为 7 时不会出现该现象.这意味着节点个数越多,系统写入负载的上限越低,这与 4.1.2 节所得出的结论是吻合的.

时间限制条件.在系统规模为 14 的条件下,我们分别设置实际写入负载为单个节点 100 Mbps、70 Mbps 和 40 Mbps.对系统任一节点的存储数据量进行监控,结果如图 7 所示.A 点、B 点和 C 点分别表示三种情况下系统累积数据处理结束的时刻.根据 4.2.1 节的结论,我们可以得到系统规模为 14 时,单个节点的写入负载上限为 70 Mbps.从图 7 我们可以得知,当实际写入负载大于该上限时,系统会在累积数据处理完成之前触发再一次的扩展;反之,则不会触发.而且,如果实际写入负载越大,那么越有可能出现处理累积数据与下一次扩展重叠的情况.导致这种现象出现的原因有两个:一是在稳定化过程中累积的数据量会随实际写入负载的增加而增长;二是在稳定化过程结束后,系统仍然需要为实际写入提供足够的带宽,越高的实际写入负载也就意味着系统可以为处理累积数据提供的带宽越少,所以处理累积数据需要的时间也越长.

在写入负载为 100 Mbps 的条件下,我们分别设置系统节点个数为 7、14 和 21.对系统任一节点的存储数据量进行监控,结果如图 8 所示.根据 4.2.1 节

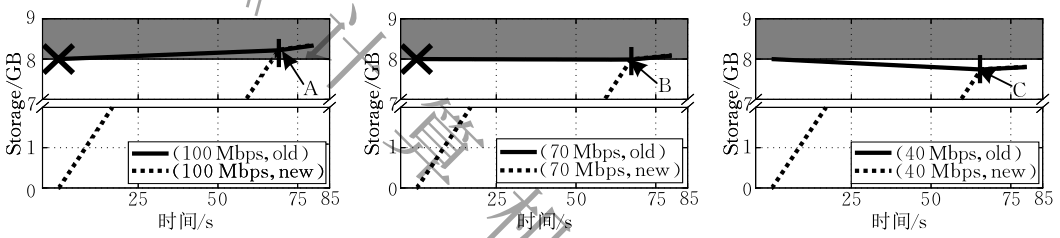

图 5 带宽限制:不同写入速率 λ(灰色区域为失效区,竖线标志位置为稳定化过程结束的时刻)

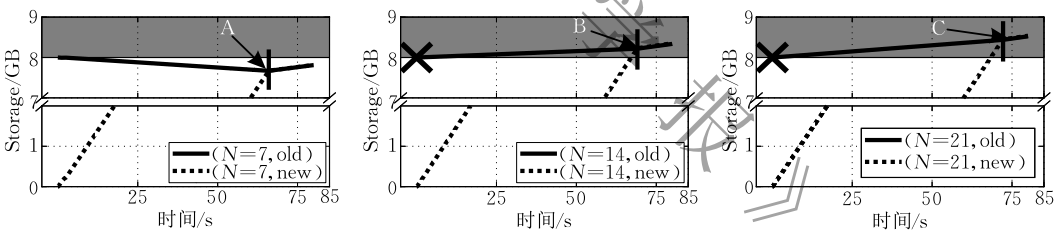

图 6 带宽限制:不同系统规模 N(灰色区域为失效区,竖线标志位置为稳定化过程结束的时刻)

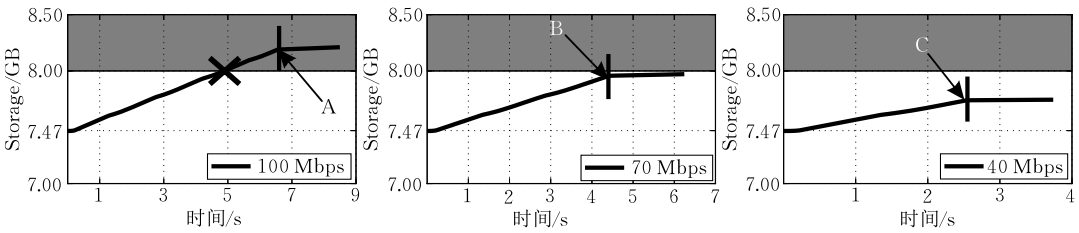

图 7 时间限制:不同写入速率 λ(灰色区域为失效区,竖线标志位置为稳定化过程中的累积写入被处理完成的时刻)

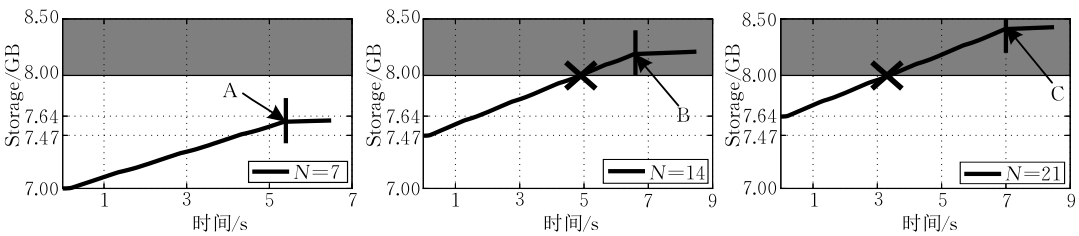

图 8 时间限制:不同系统规模 N(灰色区域为失效区,竖线标志位置为稳定化过程中的累积写入被处理完成的时刻)

的结论，当系统规模为 7 和 21 时，单个节点的写入负载上限分别为 140 Mbps 和 50 Mbps. 从图 8 我们可以得知，当系统规模为 14 和 21 时，系统在处理完累积数据之前，就已经触发了再次扩展；而系统规模为 7 时不会出现该现象. 这意味着节点个数越多，系统写入负载的上限越低，该结果印证了 4.2.1 节中的结论.

不同扩展规模. 以带宽限制为例，在写入负载为 150 Mbps，系统规模为 14 的条件下，我们分别设置新增节点个数为 1、7 和 14. 监控任一新加节点以及任一原有节点的存储数据量，结果见图 9. 根据 4.1.2 节的结论可以计算得到，当新增节点个数为 7 和 14 时，单个节点的写入负载上限分别为 500 Mbps 和 1 Gbps. 如图 9 所示，当新增 7 和 14 个节点时，系统在负载均衡完成之前，均没有触发再次扩展；但只新增 1 个节点时会出现该现象. 该结论再次验证了定理 3 的正确性.

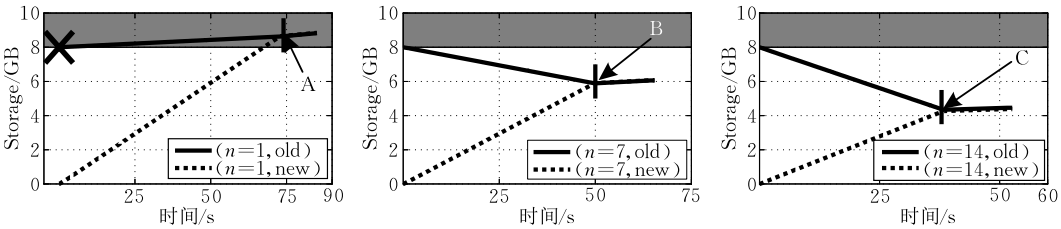

图 9　带宽限制：不同扩展规模 n（灰色区域为失效区，竖线标志位置为稳定化过程结束的时刻）

5.2　真实场景实验验证

本实验所用服务器的配置如下：CPU 为 Intel(R) Xeon(R) CPU E5-2650 v4 @ 2.20 GHz，内存为 370 GB，操作系统为 Ubuntu 16.04.5 LTS. 系统具体部署结构与参数配置考虑了中车四方案例的具体负载和应用环境信息.

5.2.1　真实场景实验验证方法与配置

真实场景实验选取了时间限制条件进行验证，针对现实中的支持负载均衡的 DHT 系统，进行了从初始化到负载再均衡，再到处理完成累积写入的全过程监控. 我们以 Cassandra 作为支持负载再均衡的 DHT 系统，以时序数据典型测试基准 Time Series Benchmark Suite① (TSBS) 作为负载生成系统.

由于我们的实验环境远好于真实应用场景的环境，所以为了更好观测实验现象，我们对写入负载以及系统单个节点的带宽加以控制，将 Cassandra 和 TSBS 封装在 Docker② 容器中，再通过 Linux 系统自带的 Traffic Control 命令对各个容器的带宽加以限制. 我们在 5 台服务器上分别放置一个封装有 Cassandra 的 Docker 容器和一个封装有 TSBS 的 Docker 容器，并开放每个容器的相关端口. 设定单个节点的带宽上限为 50 Mbps，存储空间为 13 GB，触发负载均衡的条件为单个节点数据存储量达到上限的 80%.

本实验共有以下 5 个流程：

(1) 利用 TSBS 的数据生成工具，在每台服务器的 TSBS 容器中生成 3 份数据，记为 L_1、L_2 和 L_3，分别当作系统初始化后常规的写入负载，系统处理累积写入期间的写入负载以及系统稳定化过程结束后的常规写入负载，数据量分别为 10 GB、4 GB 和 8 GB，其中原始写入负载为单个节点 30 Mbps.

(2) 启动由 4 个节点组成的 Cassandra 集群. 等到集群各节点可以正常通信之后，开始监控个节点 TSBS 容器和 Cassandra 容器的带宽使用情况，以及 Cassandra 集群的数据存储情况. 同时启动 4 个节点上的 L_1 写入程序. 需要注意的是，每个节点的 TSBS 都不会向该节点自身的 Cassandra 发送写入请求，这样做是为了保证客户端与服务端之间始终以网络通信为数据传输方式.

(3) 当某个节点的数据存储量达到触发系统负载均衡的阈值时，所有节点的 L_1 写入程序均会停止. 此时，启动第 5 个节点，并在该节点成功加入集群后，在原有的 4 个节点执行 nodetool cleanup 命令，删除转移到新加节点的数据.

(4) 在上一流程执行完毕之后，同时启动 5 个节点上的 L_2 写入程序.

(5) 在上一流程执行完毕之后，同时启动 5 个节点上的 L_3 写入程序. 等到所有写入程序完成之后，收集监控数据.

5.2.2　真实场景实验结果

在真实场景实验中，各 Cassandra 集群节点存储数据量的变化情况如图 10 所示. 其中，原点是系

① Time Series Benchmark Suite. https://github.com/timescale/tsbs/, 2020

② Docker. https://www.docker.com/, 2020

统初始化的时刻，A 点和 B 点分别是系统稳定化过程开始和结束的时刻，C 点是系统处理完稳定化过程累积写入的时刻，×表示如果没有人为干预，系统将在该时刻因为超出数据存储量限制而宕机.

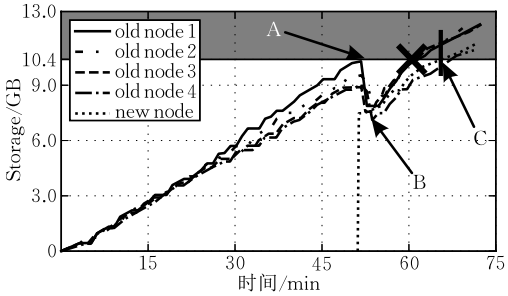

图 10 时间限制：实际系统(灰色区域为失效区，竖线标志位置为稳定化过程中的累积写入被处理完成的时刻)

根据 4.2.1 节的结论，本实验条件下系统的写入负载上限为单个节点 13 Mbps. 从图 10 可见，从原点到 A 点，系统每个节点的数据增长速率等于原始写入负载(数据增长速率为曲线的斜率)，为 30 Mbps，大于理论上限；A 点到 B 点之间，系统发生了无写稳定化过程，该过程到达的写入负载被缓存起来，等待稳定化过程结束后，再发送给系统节点；从 B 到 C 点为系统处理稳定化过程累积写入、同时接收正常写入的过程，由于需要额外处理累积写入，可以发现节点的数据增长速率要高于系统扩展前. 但是，带宽限制导致系统在处理完累积写入之前就已经触发了再次扩展的条件，即数据增长曲线进入了失效区域. 上述实验结果说明，在考虑了节点间通信等内部因素以及网络传输波动等外部因素的前提下，系统的写入负载依然会受到第 4 节所提出理论的限制.

6 应用与系统设计分析

本节首先利用第 4 节的理论结果对真实的中车四方案例进行分析，再针对当前应用较广、较知名或有一定代表性的分布式物联网数据库设计进行分析讨论.

6.1 应用案例分析

基于第 4 节的理论结果，我们对中车四方案例进行应用与分析. 首先，案例中 N 的个数为 14，而 n 的大小为 1，因而，满足理论结果的前提假设，适用于相关结论. 将 n 的数值代入后，我们可以得到：

$$\lambda < \frac{1}{N}\frac{b}{v} \tag{29}$$

即当带宽和数据单元大小确定后，系统可处理的写入负载上限与系统规模成反比.

因此，若确定系统规模为 14，假设数据单元大小为 16 字节，单个节点的带宽上限为 1 Gbps，则系统每个节点理论上能支撑的最大写入速率约为 56 万次每秒. 在实际场景中，由于消息头及消息说明等内容可能占去部分字节，因此，实际的键值大小应当大于 16 字节. 假设实际键值大小为 32 字节，则在规模为 14 的情况下，系统每个节点实际能支撑的最大写入速率约为 28 万次每秒. 如果按照案例中 480 万次每秒的写入速率进行计算，则每个节点的写入速率至少应达到 34 万次每秒. 对比理论值与实际计算值，我们发现，实际应用需求值已经超过理论允许值. 值得注意的是，这里讨论写入速率与键值大小时是分开讨论的，但在实际系统实现中，为减少通讯开销，多次写入的键值常常会合并为一次写入，但由于我们关注的是数据流量与带宽，因此实际实现的差异与理论分析本身并不矛盾. 因此，在中车四方场景中，Cassandra 在扩容之后会发生系统问题，就是由带负载均衡的 DHT 无法支撑高写入负载的理论本质导致的.

从理论分析结果与案例分析结果看来，基于 DHT 的系统，如果需要进行线上扩展过程，涉及负载再均衡，则其规模不能过大，否则，会发生整体系统问题. 但是，如果系统规模不够大，则难以承受日益增长的物联网写入负载. 因而，这构成了一对突出的矛盾. 如果从系统设计的角度来解决问题，则需要调整 DHT 的负载再均衡相关设计. 如果不采用系统负载再均衡，则系统节点容易面临部分过载，从而导致系统应用的相关问题. 因此，关于 DHT 系统设计，我们需要进一步研究如何改进，以满足即将广泛应用的物联网数据管理系统要求.

6.2 物联网数据库设计分析

物联网数据以时序数据为主体，因此也称时序数据库为物联网数据库. 以目前 DB-Engines 中排名最靠前的 10 款物联网数据库为例，进行分布式物联网数据库设计相关的讨论. 首先，考虑以 DHT 系统为基础的分布式物联网数据库，如 KairosDB，它使用的是 Cassandra，因此，在物联网数据高写入负载条件下，系统规模达到一定程度时，将发生系统问题，如在中车四方案例中发生的事件. 而 TimescaleDB 的底层存储为 PostgreSQL，其数据分布利用了类似 DHT 的原理，因此，同样存在负载再均衡过程及其

带来的严重系统问题. 而采用 HBase 的 OpenTSDB 和采用 HDFS① 的 Druid②，基于有中心节点的分布式文件系统，因而其系统存在性能瓶颈.

根据官方文档，InfluxDB 的分布式版本采用了一致性哈希，但其分布机制在系统扩展时，并不进行数据迁移和负载均衡，因此，有效避免了负载均衡 DHT 在高写入负载下的问题. 然而，InfluxDB 依然采用了 Cassandra 所用的反熵(Anti-Entropy)机制来确保数据副本间的一致性，考虑到物联网数据的海量特点，其计算比较默克尔树(Merkle Tree)所涉及的代价非常高，因此，若时序数据的保存期限很长，其引入的代价将使系统负担过重. Graphite③ 和 RRDtool④ 目前没有分布式版本，而 FaunaDB⑤ 主要是事务数据库，对时序数据的管理需求支撑有限. Kdb+⑥和 Prometheus⑦ 的分布式架构是基于网关转发设计实现的，与 DHT 原理不同，因此不会发生 DHT 扩展过程中的问题，但基于网关转发设计对于系统管理运维并不友好.

除了上述 DB-Engines 排名靠前的物联网数据库外，近期，谷歌也发表了相关文章，介绍其分布式物联网数据库 Monarch[9]. 面向高写入负载，Monarch 采用了非基于 DHT 的分片服务 Slicer[11]. 采用 Slicer 而不使用 DHT 的原因在于：(1) DHT 的无差别负载均衡难以适应冷热不均且变化不定的时序数据访问模式；(2) DHT 的无差别负载均衡无法使系统在具备充分信息的条件下做出全局最优的决策. 类似地，IBM 的相关物联网数据库[10] 也未采用基于 DHT 的可扩展架构，而是使用了共享存储，这正好与 4.3 节末尾的结论中提到的方案相符. 尽管谷歌的物联网数据库明确拒绝了基于 DHT 进行架构设计，且 IBM 的物联网数据库也未采用 DHT 架构，但是，本文第一次从理论推导的角度，给出了 DHT 无法支撑大规模物联网数据负载的根本原因.

在国内也有相关开源的分布式物联网数据库，如 TDengine⑧. 根据其白皮书⑨描述，其最近发布的分布式版本也是基于 DHT 架构设计实现的，它在系统扩展和收缩都会涉及本文所讨论的负载再均衡过程. 因此，不幸地，TDengine 存在负载均衡 DHT 在物联网数据应用场景下的所有问题，也即当系统扩展到一定规模时，TDengine 也会像中车四方案例中的 Cassandra 一样发生系统问题，对用户数据造成影响.

7　总结与展望

本文面向物联网数据管理的新场景，对经典的分布式数据管理基础设施——分布式哈希表重新进行了理论分析. 从物联网数据管理的新变化出发，即写入负载极高和带宽有限的权衡关系，本文对分布式哈希表扩展中涉及的负载再均衡过程进行了深入分析与讨论. 负载均衡对于分布式数据管理的扩展能力、负载支撑能力具有重要意义. 当系统发生变化时，负载需要再均衡. 本文的理论分析表明，尽管 DHT 能够提供非常多优越的系统属性，但支持负载均衡的 DHT 系统并不适用于写入负载会不断增加的物联网数据管理场景. 因此，对于高写入负载的物联网数据管理场景，DHT 在负载均衡方面的设计需要重新调整，否则将无法处理 IoT 场景下的高写入负载，或者会在实际应用中产生难以解决的问题.

关于 DHT 系统，从本文的理论推导结果可以推知以下两点：(1) 系统的大小可以从一开始就设置很大，因而不需要进行扩展；(2) 系统进行扩展，但不进行负载再均衡. 当满足上述条件中的任意一个时，关于 DHT 可承受写入负载的上限限制将不再存在，从而可以适用于高写入负载的物联网数据管理场景. 但是，如果条件(1)得到满足，则系统完全不具备可扩展性，也会导致资源过度消耗. 条件(2)中，如果不进行负载再均衡，系统将不再保有负载均衡特性，在实际应用中，此类系统的性能或资源利用情况也是难以令人满意的. 已知的非 DHT 方案，如 Kdb+、Prometheus 或谷歌的 Monarch[9]，都涉及多个不同功能的系统组件，相比于对称的 DHT 系统，这种不对称的系统结构增加了部署运维的难度. 什么样的系统设计才能更简洁、更好地满足高写入负载条件下的物联网数据管理需求，是需要我们未来深入探讨的新研究问题.

---

① Hadoop file system. http://hadoop.apache.org/, 2020
② Druid. https://druid.apache.org/, 2020
③ Graphite. https://graphite.readthedocs.io/en/stable/overview.html, 2020
④ RRDtool. https://oss.oetiker.ch/rrdtool/doc/rrdtool.en.html, 2020
⑤ FaunaDB. https://fauna.com/, 2020
⑥ Kdb+. https://code.kx.com/q/wp/query-routing/, 2020
⑦ Prometheus. https://prometheus.io/docs/introduction/overview/, 2020
⑧ TDengine. https://www.taosdata.com/cn/, 2020
⑨ TDengine 白皮书. https://www.taosdata.com/downloads/TDengine White Paper 20.pdf, 2020

参 考 文 献

[1] DeCandia G, Hastorun D, Jampani M, et al. Dynamo: Amazon's highly available key-value store. ACM SIGOPS Operating Systems Review, 2007, 41(6): 205-220

[2] Lakshman A, Malik P. Cassandra: A decentralized structured storage system. ACM SIGOPS Operating Systems Review, 2010, 44(2): 35-40

[3] Stoica I, Morris R, Karger D, et al. Chord: A scalable peer-to-peer lookup service for internet applications. ACM SIGCOMM Computer Communication Review, 2001, 31(4): 149-160

[4] Liben-Nowell D, Balakrishnan H, Karger D. Analysis of the evolution of peer-to-peer systems//Proceedings of the 21st Annual Symposium on Principles of Distributed Computing. Monterey, USA, 2002: 233-242

[5] Andersen M P, Culler D E. BTrDB: Optimizing storage system design for time series processing//Proceedings of the 14th USENIX Conference on File and Storage Technologies. Santa Clara, USA, 2016: 39-52

[6] Adams C, Alonso L, Atkin B, et al. Monarch: Google's planet-scale in-memory time series database. Proceedings of the VLDB Endowment, 2020, 13(12): 3181-3194

[7] Garcia-Arellano C, Storm A, Roumani D K H, et al. Db2 event store: A purpose-built IoT database engine. Proceedings of the VLDB Endowment, 2020, 13(12): 3299-3312

[8] Adya A, Myers D, Howell J, et al. Slicer: Auto-sharding for datacenter applications//Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation. Savannah, USA, 2016: 739-753

[9] Ratnasamy S, Francis P, Handley M, et al. A scalable content-addressable network//Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications. San Diego, USA, 2001: 161-172

[10] Rowstron A, Druschel P. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems //Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms and Open Distributed Processing. New York, USA, 2001: 329-350

[11] Karger D, Lehman E, Leighton T, et al. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web//Proceedings of the 29th Annual ACM Symposium on Theory of Computing. El Paso, USA, 1997: 654-663

[12] Escriva R, Wong B, Sirer E G. HyperDex: A distributed, searchable key-value store//Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication. Helsinki, Finland, 2012: 25-36

[13] Dabek F, Kaashoek M F, Karger D, et al. Wide-area cooperative storage with CFS. ACM SIGOPS Operating Systems Review, 2001, 35(5): 202-215

[14] Byers J, Considine J, Mitzenmacher M. Simple load balancing for distributed hash tables//Proceedings of the International Workshop on Peer-to-Peer Systems. Berkeley, USA, 2003: 80-87

[15] Klemm F, Girdzijauskas S, Le Boudec J-Y, et al. On routing in distributed hash tables//Proceedings of the 7th IEEE International Conference on Peer-to-Peer Computing. Galway, Ireland, 2007: 113-122

[16] Tati K, Voelker G M. On object maintenance in peer-to-peer systems//Proceedings of the 5th International Workshop on Peer-to-Peer Systems. Santa Barbara, USA, 2006

[17] Singh A. Eclipse attacks on overlay networks: Threats and defenses//Proceedings of the IEEE International Conference on Computer Communications. Barcelona, Spain, 2006

[18] Godfrey P B, Stoica I. Heterogeneity and load balance in distributed hash tables//Proceedings of the IEEE 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Miami, USA, 2005, 1: 596-606

[19] Fan B, Lim H, Andersen D G, et al. Small cache, big effect: Provable load balancing for randomly partitioned cluster services//Proceedings of the 2nd ACM Symposium on Cloud Computing. Cascais, Portugal, 2011: 1-12

[20] Cooper B F, Silberstein A, Tam E, et al. Benchmarking cloud serving systems with YCSB//Proceedings of the 1st ACM Symposium on Cloud Computing. Indianapolis, USA, 2010: 143-154

AN Yan-Zhe, Ph. D. candidate. His research interests focus on time series data management.

ZHU Yu-Qing, Ph. D., assistant research professor. Her research interests include large-scale data management and distributed system.

WANG Jian-Min, Ph. D., professor, Ph. D. supervisor. His research interests include unstructured data management, business process and product lifecycle management, digital rights, system security, and database benchmarking.

Background

Internet of Things (IoT) data management is an emerging area of research. Time series database is the dedicated database system for IoT data management. It has attracted global attention from not only academy but also industry. This is manifested by the rising popularity of time series database in the past two years, according to statistics by the famous DB-Engines website.

Due to the unprecedentedly large volume of data, the underlying scalable storage system is prominent to time series data management. Among the most popular time series databases, some have chosen the state-of-the-practice architecture of distributed hash table (DHT) for the scalable storage system, e.g., KairosDB and OpenTSDB; other state-of-the-art systems have refused the choice of DHT, e.g., Google's Monarch and IBM's DB2 Event Store. However, there does not yet exist any theoretical analysis justifying the choice or the abandon of the DHT architecture. This paper is the first attempt to address the problem theoretically.

This paper introduces this design problem of the scalable storage architecture from a real-world use case. After carefully modeling the problem as finding the feasibility conditions of DHT under the extreme IoT write workloads, this paper proves three condition bounds based on the physical constraints of storage, bandwidth and time on practical systems. These results are validated by experiments on the widely-used DHT-based system Cassandra and extensive evaluations based on a network system simulator ns-3, which has won the SIGCOMM 2020 network system award. The theoretical results are then used to study the root cause of a real-world system's problem, as well as analyzing the designs of the ten most popular time series databases on the DB-Engines website. The analysis reaches the same result as the state-of-the-art works by Google and IBM that DHT is not feasible for large-scale IoT data management scenarios. We believe this work shall lay the theoretical foundation for the design of large-scale IoT data management systems.

This work belongs to the series of work by Professor Jianmin Wang's team on IoT data management. As stated at the beginning of the paper, IoT data management is playing a more and more important role in the industry of China, especially for supporting the national strategy of *Made in China* 2025 and *New Infrastructure Construction*. In the research direction of IoT data management, Professor Wang's team is the first of all research teams in China universities to open-source an Apache Top-Level project, named Apache IoTDB.

This work addresses the fundamental problem of architecture design for IoT data management system software. The results of this paper shall assist future designs, implementations and analyses of IoT data management systems.
This work is supported in part by the National Natural Science Foundation of China under Grant No. 71690231.