清 华 大 学

综 合 论 文 训 练

题目：分布式二级哈希表

系　　别：计算机科学与技术系

专　　业：计算机科学与技术

姓　　名：冯时

指导教师：陈文光教授

辅导教师：陈康副教授

2011 年 5 月

关于学位论文使用授权的说明

本人完全了解清华大学有关保留、使用学位论文的规定，即：学校有权保留学位论文的复印件，允许该论文被查阅和借阅；学校可以公布该论文的全部或部分内容，可以采用影印、缩印或其他复制手段保存该论文。

(涉密的学位论文在解密后应遵守此规定)

签　名：__________ 导师签名：__________ 日　　期：__________

中文摘要

论文的摘要是对论文研究内容和成果的高度概括。摘要应对论文所研究的问题及其研究目的进行描述,对研究方法和过程进行简单介绍,对研究成果和所得结论进行概括。摘要应具有独立性和自明性,其内容应包含与论文全文同等量的主要信息。使读者即使不阅读全文,通过摘要就能了解论文的总体内容和主要成果。

论文摘要的书写应力求精确、简明。切忌写成对论文书写内容进行提要的形式,尤其要避免“第 1 章……;第 2 章……;……”这种或类似的陈述方式。

本文介绍清华大学论文模板 ThuThesis 的使用方法。本模板符合学校的本科、硕士、博士论文格式要求。

本文的创新点主要有:

- 用例子来解释模板的使用方法;
- 用废话来填充无关紧要的部分;
- 一边学习摸索一边编写新代码。

关键词是为了文献标引工作、用以表示全文主要内容信息的单词或术语。关键词不超过 5 个,每个关键词中间用分号分隔。(模板作者注:关键词分隔符不用考虑,模板会自动处理。英文关键词同理。)

关键词: TEX　LATEX　CJK　模板　论文

ABSTRACT

An abstract of a dissertation is a summary and extraction of research work and contributions. Included in an abstract should be description of research topic and research objective, brief introduction to methodology and research process, and summarization of conclusion and contributions of the research. An abstract should be characterized by independence and clarity and carry identical information with the dissertation. It should be such that the general idea and major contributions of the dissertation are conveyed without reading the dissertation.

An abstract should be concise and to the point. It is a misunderstanding to make an abstract an outline of the dissertation and words "the first chapter", "the second chapter" and the like should be avoided in the abstract.

Key words are terms used in a dissertation for indexing, reflecting core information of the dissertation. An abstract may contain a maximum of 5 key words, with semi-colons used in between to separate one another.

Key words: TEX LATEX CJK template thesis

目　录

第 1 章　引言

1.1　研究背景

分布式存储系统越来越成为整个互联网赖以存在和发展的强大依托。从 20 世纪 90 年代初互联网兴起开始,我们进入了一个信息爆炸的时代。2002 年世界上共产生了五百亿亿字节的数据,其中 92% 的信息存储于电子介质中,这相当于人类历史上所有说过的话语所包含的信息量的总和。[1] 随着互联网的进一步发展和普及,信息产生的速度还在加快。单一的存储单元已经不能容纳如此巨大的信息量,分布式存储系统通过将多个存储单元组织起来,充分利用每个存储单元的资源,使得整个系统的存储容量成倍的增长,可以很好的满足大容量存储的需求。另一方面,随着云计算的迅猛发展,互联网用户希望更多的个人数据存储在云端,而不是本地个人计算机上面。分布式存储系统为云计算和云存储提供强大的后端支持,在满足海量个人数据存储的同时,使得用户不必关心数据的组织方式和存储实现。

存储系统归根结底解决的是从索引到值的映射关系,分布式哈希表是最基本的分布式存储系统。任何一种分布式存储系统归根结底都是一种广义上的分布式哈希表。只不过在这样的系统中,索引可能是比字符串更复杂的数据结构。例如在像 Google File System[2] 这样的分布式文件系统中,数据被看作文件。分布式文件系统按照层级目录的方式组织数据,用户则通过指定完整路径来索引文件,并通过读、顺序写等基本文件操作来存取数据。另一类分布式存储系统则通过支持更复杂的值类型来提供更丰富的语义。例如在分布式存储系统 PNUTS[3] 中,存储的对象可以是字符串、整数等基本数据类型,也可以是没有子结构和含义的二进制数据块①。此外,分布式数据库还维护了数据之间的关系,支持简单的查询语义。

分布式存储系统的根本设计原则之一,是根据上层应用的需求,在系统的简洁性和功能的丰富性之间找到一个最优平衡点。不考虑设计者的因素,功能越强大,系统势必越复杂庞大,系统的运行效率可能越低。Dynamo[4] 等经典分布式

① http://en.wikipedia.org/wiki/Binary_large_object

系统不止一次阐明了这个原则:一个分布式系统不是具备越丰富的功能越出色,而是能够高效率实现足够的功能。具体到分布式存储系统,在相同实验条件下,最基本的分布式哈希表支持的语义最简单,执行效率也最高。与之相比,分布式文件系统和分布式数据库等存储系统功能更强大,方便上层应用调用,但运行效率相对较低。

随着存储成本的降低、网络条件的改进以及安全技术的发展,云存储越来越受到人们的青睐。更多的用户希望把个人数据存放在互联网上,以增强数据存储的可靠性,节省本地存储空间,实现数据的离线传输,以及方便不同终端之间数据同步。此类应用在互联网上也层出不穷,比如以 Gmail① 为代表的电子邮件系统,以 flickr② 为例的个人在线图片存储系统等,都能满足用户在线存储数据的需要。这些应用需要存储的数据具有共同的特点:数据是按照用户分开存储的;每个用户可以存储多个数据;不要求系统维护数据之间的关系。

针对上述云存储应用的特点,我设计并实现了分布式二级哈希表。该存储系统提供简洁而易用的接口,可以作为这些应用的底层存储后端。在一个分布式二级哈希表中,数据是二进制块,对于数据的索引通过指定二级 ID 实现。这样,第一级 ID 可以用于区分不同用户,第二级 ID 则用来指定同一个用户的不同数据。数据则根据第一级用户 ID 分配到不同的存储服务器上,实现分布式存储。为了保证系统的运行效率,我在实现分布式二级哈希表的过程中借鉴并使用了部分开源代码。其中,每台存储服务器上都部署了 Redis ③ 实现本地哈希表存储,数据分布和备份采用了著名的一致性哈希算法[5],基于开源项目 libconhash ④ 作出修改实现。

1.2 相关工作

1.2.1 分布式哈希表

分布式哈希表是最基本的分布式存储系统。单机哈希表解决的是从索引到值的映射关系,而分布式哈希表则是根据索引将不同的数据分配到不同的存储服务器上,从而实现数据的分布式存储。一般方法是:先确定一个哈希函数,将此

---

① http://mail.google.com/mail
② http://www.flickr.com/
③ http://redis.io/
④ http://sourceforge.net/projects/libconhash/

哈希函数的值域空间按照某种方式分割成多个子空间,每一个子空间对应一台存储服务器。当我们需要确定某个数据在哪台服务器上存放时,就把这个哈希函数作用在它的索引上,得到的哈希值所在的子空间对应的服务器上就存储了或者应该存放此数据。由于分布式哈希表原理简单,实现一个高效率的系统并不难。现在已经有很多成熟的实现,比如豆瓣的 BeansDB ①,亚马逊的 Dynamo 等分布式哈希表,都具有很高的执行效率。

使用基本的分布式哈希表难以实现高效率的分布式二级哈希表。由于单个一级 ID 可能对应多个二级 ID,使用基本分布式哈希表很难罗列出某个一级 ID 对应的所有二级 ID 以及数据。我们希望能够实现一个原生支持二级哈希语义的系统,既要保证操作的原子性,又能够高效率运行。

1.2.2 分布式文件系统和分布式数据库

Google File System 等分布式文件系统和以 Bigtable[6] 为代表的分布式数据库支持比分布式哈希表更复杂的语义。在分布式文件系统中,数据被看作文件,通过路径来索引。分布式数据库则侧重数据之间的关系,支持一些匹配查询。这些系统在处理大块数据时表现出很好的性能,但在操作小数据(小于 1MB)时,系统开销则相对过大。在这种情形下,采用简单的分布式哈希表来处理小规模数据更合适一些。[4] 虽然分布式文件系统和分布式数据库可以实现分布式二级表的全部语义,但是我们希望设计一种更轻量级的系统,它在处理小数据的时候能够表现出很高的性能。

① http://code.google.com/p/beansdb/

插图索引

表格索引

公式索引

参考文献

[1] 马少平, 刘亦群. 去伪存真, 去粗取精——页面质量评估及其在网络信息检索中的应用. 2006

[2] Ghemawat S, Gobioff H, Leung S. The Google file system. Proceedings of ACM SIGOPS Operating Systems Review, volume 37. ACM, 2003. 29–43

[3] Cooper B, Ramakrishnan R, Srivastava U, et al. PNUTS: Yahoo!'s hosted data serving platform. Proceedings of the VLDB Endowment, 2008, 1(2):1277–1288

[4] Hastorun D, Jampani M, Kakulapati G, et al. Dynamo: amazon's highly available key-value store. Proceedings of In Proc. SOSP. Citeseer, 2007

[5] 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 Proceedings of the twenty-ninth annual ACM symposium on Theory of computing. ACM, 1997. 654–663

[6] Chang F, Dean J, Ghemawat S, et al. Bigtable: A distributed storage system for structured data. ACM Transactions on Computer Systems (TOCS), 2008, 26(2):1–26

[7] MooseFS. http://www.moosefs.org/

[8] BeansDB. http://beansdb.googlecode.com/files/Inside%20BeansDB.pdf

[9] Kyoto Cabinet. http://fallabs.com/kyotocabinet/kyotoproducts.pdf

[10] Chu S. MemcacheDB. http://memcachedb.org/memcachedb-guide-1.0.pdf

[11] Citrusbyte. Redis. http://redis.io/

附录 A　外文资料的的调研阅读报告

A Brief Report on Distributed Storage System

Introduction

Distributed system is a major topic of current computer science. Among it, a distributed storage system usually takes advantage of multiple machines to gain capacity, reliability, availability, while for people who operates data, it seems that he is facing with a single machine and needn't care about the whole backend framework.

Categories

There are a lot of typical distributed storage systems. Google File System[2], MooseFS[7] and many others are classified as *distributed file system*. A client can communicate with the cluster to store and retrieve data much like operating files in a local file system. Usually, they support full namespace hierarchy and the data is accessed with a path.

Another category is classified as *distributed database*. The significant feature of the data stored in a distributed database is that the data is stored and accessed aligned by *columns* and *rows*. Sometimes, more strengthful databases also record relationship between data and support complicated query semantics.

Apart from the two categories mentioned above, a novel kind of distributed storage system is getting more and more popular. It is not only light-weighted, as it doesn't support relationship between data or just supports weak relationship, but also flat, usually because it doesn't support complicated namespace hierarchy but just a map of key-value pairs. On the opposite, the system which belongs to this category is usually of high performance, to be measured by throughput, latency, availability, reliability and other criteria. Such systems are called *distributed key-value stores*. Douban's BeansDB[8], Kyoto Cabinet[9] from Japan and many others are all successful distributed key-value stores. As such storage systems are light-weighted, some systems, like MemcacheDB[10], decide to put data in memory or virtual memory to gain performance burst.

Fundamental Concepts

Before I summarize existing distributed storage systems, I would like to explain some fundamental concepts related to distributed system. Without a clear introduction on these conceptions, it would be impossible to comprehend the beauty and elegance of architecture designs on famous distributed storage system.

Replication

Replication is copies of same data. In distributed system, data is distributed to many computing and storage machines. Under most cases, data is evenly stored on different machines. The world will be easy but fragile if we don't make copy of data. Let's take a closer look at why data backup is necessary with some simple calculation. If the possibility of failture for a single machine is p, and for simplicity, we assume all the machines are independently identical, i.e., the possibility of failure for each machine equals p and a machine never notices whether his buddy is alive or dead. What is the possibility that a system containing n identical machines goes down with one or more machine failing to work at some time? Yes, you are right! It is $1-(1-p)^n$. So what does this mean? Although p may be small, when n gets larger and larger, the result tends to reach 1! Commonly, a datacenter of companies like Google contains from thousands of to millions of machines with moderate disks. Disk failures are not rare but a common case. If the data is not replicated, it would be impossible to ensure the integrity of data, as recovery of data from a failed disk costs time, computation resource and network bandwith, yet this is not always possible. Hence making copies of data is critical in large systems like distributed key-value store.

Consistent Hashing

Replication of data may not be as simple as it seems at first glance. There are many tricky technologies behind to provide the correctness of the replication mechanism, and further the improvement of the performance. Situation becomes even worse when the system is distributed as we have to make decision on the choice of machine for different block of data.

Usually, the data is distributed by its key. First, a hash function is needed to calculate a hash value for the key. The key may be a string or even any binary stream, while the result of the hash function always falls into a finite range. The range, called *hash space*, is divided into several segments, each representing a machine. That machine takes responsibility of all the data with keys whose hash value is within that range.

Different distributed algorithms vary in their hash functions. A simple instance could be a distributed system which adopts *Cyclic Redundancy Check* as its hash function. As we mentioned above, each machine is associated with a segment in the range of hash value. However, in most cases, this range is further made up of several subsegments, called hash slots. Often, hash slots within a segment are not located adjacently to each other in the hash space. Note that this is why the algorithm distributes data evenly among different machines, while the impact of a single machine failure is minimized.

Consistent hashing is a method to distribute data evenly within a storage cluster, regardless of the specific hashing function employed. If we concatenate the tail of the hash space to its head, the hash space will rewind like a ring, called the *hash ring*. We put some nodes on the ring and the ring is broken up into some arcs. These nodes are called *virtual nodes*. A physical node is a storage server, which is a collection of same amount of virtual nodes. The assignment of virtual node to physical one is random, which means that all the virtual nodes, which belong to the same physical node, may not be adjacent on hash ring. Different virtual nodes, which belong to different physical nodes, may not appear on the hash ring in a round-robin manner. They are just randomly distributed.

How to decide which storage server should take responsibility of the data associated with a specific key? If we don't take data replication into consideration, the server is the physical node found as follows. We traverse clockwise from the point on the hash ring representing the hash value of that key. The physical node, where the first virtual node met like this belongs to, is the storage machine which is responsible for that data.

When a machine refuses to work any more due to failure, the virtual nodes associated with it should be taken care by other physical nodes. It is obvious that each of such virtual nodes should share the same physical node, to which the next virtual node belongs, counting clockwise. As the virtual nodes conducted previously by the failed machine are distributed randomly, it is probably expected that the data handled previously by the failed machine will be distributed evenly across other active machines. It goes the same when a new machine joins the cluster.

In the real world, data is replicated into many copies. The data associated with a single key is stored on several machines, namely N physical nodes. These machines are determined by tranversing clockwise from the point on the hash ring representing the hash value of that key. The traversal stops when the first M virtual nodes belongs to N different physical nodes. These N physical nodes are the target machines.

Example Distributed Storage Systems

In this short article, I'll make a brief summary on several famous distributed storage system with introductions on their significant features.

Dynamo: Amazon's Highly Available Key-value Store

I believe the most famous distributed key-value store is Amazon's Dynamo[4]. Although it is neither open source nor available for organizations outside Amazon to use, it is highly recognized by scientists and engineers in the area of distributed computing.

Perhaps Dynamo gains popularity mainly because of its elegant design. It is a perfect example of minimizing system functionality to satisify basic requirements of application. Dynamo acts as an internal infrastructure for Amazon's many services, such as the on-line book stores. In most of the senarios, the service beyond Dynamo has such a requirement that data is highly writable. Considering this specific requirement, Dynamo is designed at first day to support high throughput and low latency of write request, while to sacrifice consistency hence increasing of read request both operation time and possibility of version conflict, which is tolerable within these services.

Bigtable: A Distributed Storage System for Structured Data

Bigtable[6] is an outstanding representative of the brand new *NoSQL*. It differs from traditional database by not supporting complex relationship between data, yet more flexible. Bigtable is considered as a multi-dimensional mapping of data. The last dimension is usually timestamp which means the database records historical snapshots of data. The database is flexible that dimension names of different data may be different.

Redis: An Open Source, Advanced Key-value Store

Redis[11] is an open source key-value store. It beats other hash tables for its rich type of values, such as binary stream, lists, sets, sorted sets and even hashes, while still maintaining high performance because all the data resides in memory. Redis not only performs data compression but also implements a virtual memory layer in user space to solve memory shortage. It is also fully journaled to enhance ability of fault tolerance.