云计算概述

云计算，作为长期以来梦寐以求的计算方法，有使 IT 行业发生巨大改变的潜力，使软件作为一种服务更加具有吸引力，并塑造了 IT 硬件的设计和购买方式。对新互联网服务有创新灵感的开发者不再需要在硬件上花费大量资金来部署他们的服务，或在运作上花费大量的人力成本。他们不需要担心在服务的流行程度低于预期的情况下，进行了过多的准备，浪费了昂贵的资源；或者是当服务非常流行时，准备不足，造成潜在客户和收入流失。更重要的是，需要处理大数据的公司能尽快得到处理结果，就像程序能自动调整一样，因为 1000 台服务器运行 1 小时的开销不多于 1 台服务器运行 1000 小时的开销。不需要额外开销就能获得更大的规模，这样的资源弹性在 IT 行业中是史无前例的。

因此，云计算在博客和官方报道中是一个非常热门的话题，并且出现在研讨会、会议甚至是杂志的标题之中。然而，人们对云计算到底是什么和在什么情况下能发挥作用仍存在很大的困惑，因为 Oracle 的 CEO Larry Ellison 沮丧地说过：“云计算的有趣之处在于我们重新定义了云计算，使它包含了一切……我不明白除了改变了一些广告用语之外，我们用云计算做了些什么不同的事。”

本文的目的是解释一些术语，提供简单图表定量地比较云计算和传统的计算，以及鉴别云计算在技术和非技术方面上的挑战和机遇，通过这些减少人们对云计算的困惑。

1 定义云算

云计算涉及互联网服务应用和提供这些服务的数据中心的硬件和系统软件两个方面。这些服务就事是人们所熟知的软件即服务（SaaS，Software as a Service）。一些厂商使用术语，比如设施即服务（IaaS，Infrastructure as a Service）和平台即服务（PaaS，Platform as a Service）来描述他们的产品，但我们避开这些定义还存在很大分歧的话题。“低层次”的设施和“高层次”的平台之间的界线划分已经不是什么新鲜话题。我们相信这两者是相似的，而不是不同的，并且我们把他们当作一个整体进行考虑。相似地，来自高性能计算社区的相关术语“网格计算”，提倡提供共享计算和远距离存储的协议，但这些仅在它的社区范围内有效。

数据中心的硬件和软件就是我们所说的云。当云以即用即付费的方式面向普通大众时，我们称它公共云；所出售的服务是效用计算。而私有云指那些不向普通大众开放的企业或组织内部的数据中心，同时这些数据中心必须有足够大的规模，使企业或组织能从我们这里讨论的云计算的优势中获利。这样，云计算是软件服务和效用计算的结合，但并不包括那些中小型的数据中心，即使他们依赖虚拟化管理。人们可能是软件服务的用户或提供商，或效用计算的用户或提供商。我们主要关注和软件用户相比不

那么引人注目的软件服务提供商（即云用户）和云提供商。图 1-1清楚的表示了提供商和用户之间的关系。在某些情况下，一个人可以扮演多个角色，一个云服务提供商在云设备中也可能拥有它自己的面向客户的服务。

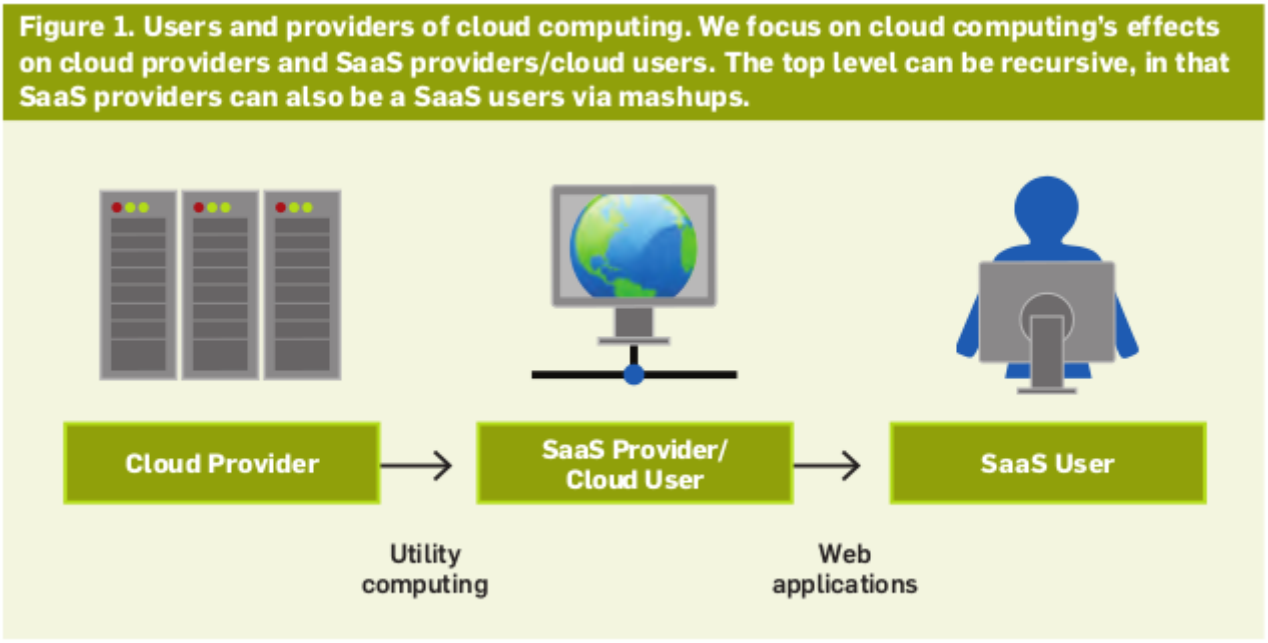

图 1-1　云用户和云提供商

从硬件准备和价钱的角度来看，云计算有三个新的特点。

- 在需要的时候能提供无限的计算资源，能快速应对负荷冲击，因此消除云计算用户需要计划长远和事先准备的忧虑。
- 云用户不必交纳预付费用，因此允许公司从小开始，并且只有在需求快速增长时才增加硬件资源。
- 短期内，能够按照云计算资源的使用需求进行付费（例如：处理器按小时计费，存储容量按天计费），并且当在需要的时候释放这些资源，因此能够通过释放不再需要的机器和存储空间达到节约资源的目的。

我们证明极大规模的、商用电脑计算中心建设和运作在低成本的地点是云计算的关键、必须的要素，这在很大程度上减少了 5 到 7 倍在电力、网络、带宽、运作、软件和硬件上的成本。这些因素，结合用来增加利用率的统计复用，意味着和传统的数据中心相比，云计算能提供成本低于中型数据中心的服务，并且还能带来不错的利润。

我们提出的定义让我们能够清楚地认识某些例子是属于云计算还是非云计算。一个托管在因特网服务提供商的面向公众的因特网服务，需要为服务分配更多机器时需要 4 个小时的时间。但在公共因特网上的负荷冲击发生的更加迅速（Animoto 的负载连续了 3 天每 12 小时翻一倍），所以这不是云计算。相反地，一个内部的企业数据中心，它的应用只有在显著提前通知管理员时才被修改。在这种情况下，分钟级的大规模

的负荷冲击不太可能发生，这种情况符合作为云计算的必要条件之一。但这个企业的数据中心可能仍然不满足作为云的其他条件，比如无限的资源或者细粒度的账单。一个私有的数据中心可能也不能从使公共云在财务方面具有吸引力的经济规模中获利。

忽略私有云导致了博客中相当多的争论。我们相信当中型数据中心也声称有公有云的优势时，将会产生 Larry Ellision 的引用所表达的困惑和怀疑。除了拥有成百上千台机器的极大的数据中心，例如像 Google 和 Microsoft 运营的数据中心，大多数的数据中心只享用了部分公有云的潜在优势，如图 1-2所示。因此我们相信将传统的数据中心包括在云计算的定义范围内将导致夸大小型的、所谓的私有云，这也是我们不把它们包含在内的原因。然而，在这我们描述了所谓的私有云如何通过超负荷计算或是混合云计算获得更多公有云的利益。

Table 1. Comparing public clouds and private data centers.

Advantage	Public Cloud	Conventional Data Center
Appearance of infinite computing resources on demand	Yes	No
Elimination of an up-front commitment by Cloud users	Yes	No
Ability to pay for use of computing resources on a short-term basis as needed	Yes	No
Economies of scale due to very large data centers	Yes	Usually not
Higher utilization by multiplexing of workloads from different organizations	Yes	Depends on company size
Simplify operation and increase utilization via resource virtualization	Yes	No

图 1-2　公有云和私有数据中心对比

2　效用计算分类

任何应用都需要计算模型、存储模型和交互模型。统计复用必须要实现弹性和根据需要自动分配和管理无限的容量的功能。实践中，这些通过虚拟化实现。我们的观点是不同的效用计算服务将基于不同的云系统软件的抽象层次和资源的管理层次进行区分。

Amazon 的 EC2 在这个范围的一端。一个 EC2 实例看起来很像一个物理硬件，且从内核启动后，用户可以控制几乎整个软件栈。这种低层次使 Amazon 难以提供自动化的可伸缩性和失效备援，因为和响应以及其他状态管理问题有关的语义很大程度上是应用相关的。在这个范围的另一极端，是像 Google 的 AppEngine 一样的应用领域特

定平台，专门针对传统的网页应用，迫使应用的架构清楚地分离无状态的计算层和有状态的存储层。AppEngine 令人印象深刻自动调整和高可用性的机制，和 AppEngine 应用可用的专有大数据存储都依赖于这些约束。Microsoft 的 Azure 应用是用.Net 库编写，并编译成一个语言相关的管理环境—公共语言运行库。这个框架明显地比 AppEngine 的更灵活，但仍然在存储模式和应用架构方面限制了用户的选择。因此，Azure 是处于像 AppEngine 的应用框架和像 EC2 这样的硬件虚拟机之间的位置。

3 云计算经济学

我们来看三个特别地引人关注的，相比传统主机更青睐效用计算的用例。第一个例子是在需要的服务随时间而改变的情况下。比如，为高峰负荷准备一个数据中心，这必然能够承受每月那么几天的高峰，然后却也导致在其他时间得不到充分利用。取而代之，云计算让一个组织按小时支付使用的计算资源，这样能有效地节约成本，即使从云提供商那租用的机器每小时的费用比购买一台机器贵。第二个例子是在需求不明的情况下。比如：一个刚搭设的网站当它变得流行起来时，也潜在伴随着访客离开减少的可能性，这时它需要能支撑高峰。最后，进行批量分析的组织能够使用云计算的成本关联性更快地完成计算：使用 1000 台 EC2 的机器一小时的开销和使用 1 台机器 1000 小时的开销一样。

虽然云计算在经济上的吸引力经常被描述为“将资金开销转变为运营开销”（CapEx to OpEx，converting capital expenses to operating expenses），但我们相信“即用即付费”更加直接地抓住了买家的经济效益。通过云计算购买的小时数能按时间上非均匀的方式分配（例如，今天使用 100 小时而明天不使用，则只需支付 100 小时的费用）；在网络社区中，这种销售带宽的方式是已被熟知的基于流量的定价。除此之外，消除了预付资金开销能让资金被重新被使用在核心业务的投资上。

因此，即使在相同时期，Amazon 的即用即付费定价比购买和贬值一台类似的服务器贵，我们认为极端重要的云计算的弹性和风险转移带来的经济效益，尤其是过度准备（未充分使用）和准备不足（达到饱和）的风险，比成本更加重要。

我们从弹性开始说起。主要观察到云计算能以良好的粒度增加或移除资源的能力（EC2 每次一台服务器）和分钟级的交付周期而不是星期级能让资源和工作量匹配地更加接近。世界上数据中心的服务器的平均利用率从 5% 到 20%。这听起来低得令人吃惊，但符合许多服务的工作高峰超过了平均值 2 到 10 倍的观测。因为几乎没有用户故意做少于预期峰值的准备，资源在非高峰时段处于闲散状态。变化越显著，浪费就越多。

例如，图 3-1(a）假设我们的服务有在中午的峰值时需要 500 台服务器的预测需求，但是在半夜的波谷时只需要 100 台服务器。只要一整天的平均利用率是 300 台服务器，

每天实际的成本（在这个曲线之下）是 300×24=7,200 服务器小时；但是因为我们必须为峰值准备 500 台服务器，我们支付 1.7 倍的费用 500×24=12,000 服务器小时。因此，只要即用即付费每服务器小时的花费在三年内（典型的折旧时间）少于 1.7 倍的购买服务器的费用，那么效用计算就相比之下较便宜。

实际上，这个例子低估了弹性带来的效益，因为在除了简单的每日模型之外，大多数服务也经历过季节性的或其他周期性的需求变化（例如：电子商务在 12 月份和照片分享网站在假期后访问量急剧上升），还有一些因为外部事件的不可预期的需求爆发（例如：新闻事件）。因为可能要花上几周的时间来获得和布置新设备，所以你必须为这些峰值事先做准备。我们已经看见即使服务运营商正确地预测了峰值的规模，但资源却被浪费了，并且如果他们的准备低估了峰值，那可能会更糟。

他们可能也会低估如图 3-1(b）所示的峰值，这会意想不到地使过量用户离开。过度准备的开销容易衡量，而准备不足的开销却难以衡量，且有同样的潜在严重性：不仅拒绝用户不产生收入，并且这些用户可能永远流失了。例如：Friendster 的流行度相对竞争对手 Facebook 和 MySpace 有所下滑，部分原因归咎于用户对缓慢的响应时间（最多 40 秒）的不满。图 3-1(c）旨在捕获这种行为：用户会抛弃一个准备不足的服务直到高峰用户负载等于数据中心正常的负载能力，这时用户才能获得能够接受的服务质量。

为了简化这个例子，假设一个虚拟的网站的用户可分为两类：活跃用户（经常访问该网站的）和流失用户（抛弃了这个网站或由于网站差劲的表现离开的）。进一步假设 10% 的活跃用户在获得由于准备不足导致的差劲服务后永远流失了（成为流失用户），这样，留下来的常规用户就会获得更好的体验。这个网站起初准备应对 400,000 用户的预期峰值（每台服务器应对 1000 用户 ×400 台服务器），但意料之外的正面报道在第一个小时内带来 500,000 的用户访问量。在 100,000 位离开或获得差劲服务的用户里，根据我们的假设他们当中的 10,000 位永远流失了，留下了 390,000 位活跃用户。接下来的一小时中又有 250,000 位新用户访问。起初的 10,000 位用户获得了良好服务，但是仍然有 240,000 位用户超出了网站的容量。这导致增加了 24,000 位流失用户，剩下 376,000 的固定用户。如果这个模式持续下去，在 lg(500,000) 或 19 小时之后，新用户的数量将接近 0，并且网站的容量将达到稳定。显然，在这 19 小时中，网站运营商获取了少于与 400,000 用户等值的稳定收入，而这再次说明了不足之处，更不用说不满用户留下的坏名声。

这些情况真的会发生在实践之中吗？当 Animoto 通过 Facebook 提供服务时，它遭受了导致在 3 天之内从 50 台服务器增长到 3500 台服务器的需求冲击。即使每台服务器的平均利用率低，没人可以预见对资源的需求会突然连续三天每 12 小时翻倍。在高峰之后，通信量下降到一个较低的水平。所以在这个真实的案例中，可调整的弹性不是成本优化，而是一个运作要求，并且可下调减少的弹性能允许在稳定状态时的开销

更加接近符合稳定状态的工作量。

Figure 2. (a) Even if peak load can be correctly anticipated, without elasticity we waste resources (shaded area) during nonpeak times. (b) Underprovisioning case 1: potential revenue from users not served (shaded area) is sacrificed. (c) Underprovisioning case 2: some users desert the site permanently after experiencing poor service; this attrition and possible negative press result in a permanent loss of a portion of the revenue stream.

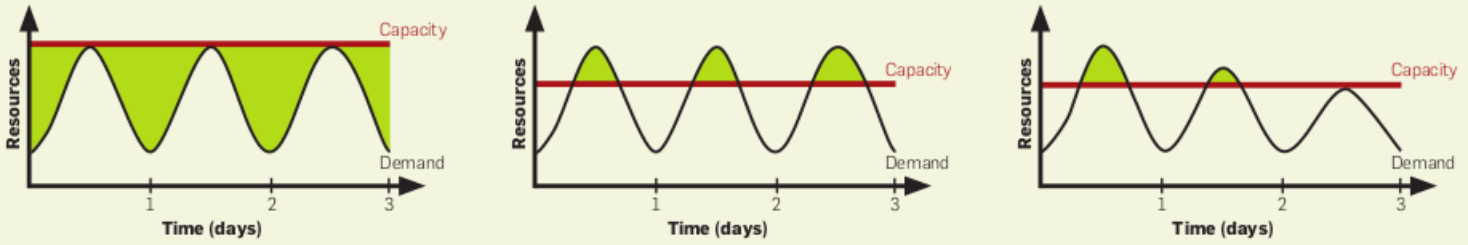

图 3-1

4 云计算的 10 个障碍和机会

图 4-1总结了阻碍云计算发展的决定性的障碍的排名列表。前三个影响应用，后五个影响增长，最后的两个是政策和商业障碍。每个障碍都伴随着一个克服障碍的机会，包括从产品发展到研究项目。

4.1 商业持续性和服务可用性

一些组织担心效用计算是否有足够的可用性, 这造成了一些对云计算的警觉。讽刺的是，现有的软件服务产品在这点上设立了一个很高的高标准。Google 搜索以高可用性著称，在这点上甚至一个小小的崩溃都会被主要新闻媒体争相报道。

用户期望新服务能有相似的可用性，但这很困难。图 4-2展示了 Amazon 简单存储服务（S3，Simple Storage Service），AppEngine 和 Gmail 在 2008 年的服务中断记录。即使这些中断造成了负面的影响，但几乎没有企业的 IT 设施像他们这么好。除了可用性的技术问题，云提供商可能因非技术问题遭受服务中断，包括停业或者成为调控活动的目标（后者最近的一个例子发生在去年，我们稍后会描述）。

云提供商能提供专门的硬件和软件技术来提供更高的稳定性，但他们却没这么做，大概是因为价钱太高。这样的稳定性之后可以作为服务水平协议出售给用户。但是这种方法只能这样。高可用性计算社区已经长时间保持了“不败记录”的美誉，然而单一公司管理云计算服务就是一个失败。即使公司在不同地理地区拥有多个数据中心，并使用不同的网络提供商，但仍然会有常见的软件设施和会计系统，或者公司甚至停业。

Table 2. Top 10 obstacles to and opportunities for growth of cloud computing.

	Obstacle	Opportunity
1	Availability/Business Continuity	Use Multiple Cloud Providers
2	Data Lock-In	Standardize APIs; Compatible SW to enable Surge or Hybird Cloud Computing
3	Data Confidentiality and Auditability	Deploy Encryption, VLANs, Firewalls
4	Data Transfer Bottlenecks	FedExing Disks; Higher BW Switches
5	Performance Unpredictability	Improved VM Support; Flash Memory; Gang Schedule VMs
6	Scalable Storage	Invent Scalable Store
7	Bugs in Large Distributed Systems	Invent Debugger that relies on Distributed VMs
8	Scaling Quickly	Invent Auto-Scaler that relies on ML; Snapshots for Conservation
9	Reputation Fate Sharing	Offer reputation-guarding services like those for email
10	Software Licensing	Pay-for-use licenses

图 4-1 云计算的机会和阻碍

在这种情况下没有商业可持续性战略，使大量的客户不愿意迁移到云计算。我们相信独立软件栈最好的机会就是由不同的公司提供，因为一家公司很难以可靠性的名义来建设和维护两个栈。就像大型因特网提供商用多个网络提供服务，这样一家公司的失败不会使他们全部下台，我们相信要达到高稳定性，唯一合理的解决方案就是多个云服务商。

Table 3. Outages in AWS, AppEngine, and gmail service and outage duration date.

Service and Outage	Duration	Date
S3 outage: authentication service overload leading to unavailability[17]	2 hours	2/15/08
S3 outage: Single bit error leading to gossip protocol blowup[18]	6–8 hours	7/20/08
AppEngine partial outage: programming error[19]	5 hours	6/17/08
Gmail: site unavailable due to outage in contacts system[11]	1.5 hours	8/11/08

图 4-2 AWS、AppEngine 和 Gmail 服务器崩溃情况

4.2 数据锁存

软件栈改进了平台之间协同工作的能力，但是云计算的存储 API 本质上还是专用的，还不是标准化的。因此，客户不能轻易地从一个网站中提取出它们的数据和程序然后在另一个上运行。从云中提取出数据的难度阻碍了一些组织采用云计算。客户锁存可能对云提供商来说有吸引力，但他们的用户却易受提价、可靠性问题、甚至提供商停业的危害。

例如：2008 年 8 月 8 日，一个叫作 Linkup 的在线存储服务在丢失了 45% 的用户数据后宣告倒闭。Linkup 反过来依赖在线存储服务 Nirvanix 来存储客户数据，两个组织之间的争端导致了客户数据的丢失。同时，20,000 位 Linkup 的用户被告知服务不再可用并催促他们尝试其它的存储网站。

一种解决方案是 API 标准化，通过这种方式软件服务开发者就能够把他们的服务和数据部署在多个云计算提供商上，这样一家公司的失败不会造成所有用户数据拷贝的丢失。可能有人会担心这会引发云服务的价格战，比且减少云计算提供商的利润。我们提供两个论据来缓和这种恐惧。

首先，服务质量和价钱一样重要，所以顾客可能不会选择价格最低的服务。所以今天的一些因特网提供商的费用是其他提供商的 10 倍，因为他们更加可靠，且提供额外的服务来提高可用性。

其次，除了缓解数据锁存的顾虑，标准化 API 能带来新的使用模式，相同的软件设施既能在内部数据中心，也能在公共云上被使用。这样的选择性能使公共云用于捕获那些由暂时的高负载引起的，不能轻易在数据中心（私有云）中运行的额外任务，从而实现混合云计算或者超负荷云计算。这种选择性可以显著地扩大云计算的市场。确实，专用云的 API 实现开源，例如 Eucalyptus 和 HyperTable，是实现超负荷计算的第一步。

4.3 数据保密/可审计性

尽管大多数公司薪酬外包，且很多公司使用内部邮箱来保存敏感信息，但安全问题是一个最经常被引用来反对云计算的话题。分析师和持怀疑态度的公司问道：“谁会信任把他们的重要数据放在外面?”并且还需要可审计，从健康和人类服务萨班斯 -奥克斯利法案和健康保险流通与责任法案规定的意思上看，必须为企业提供将数据迁移到云的服务。

云用户面对着来自云外部和云内部的危险。大多数安全问题都涉及到保护云免受那些大型数据中心同样要面对的相似的外部威胁。然而在云中，可能很多的群体都要承担这项责任，包括用户的安全敏感的软件或配置所依赖的云用户，云提供商和任何的第三方提供商。

用云户需要承担应用层的安全。云提供商需要承担物理层的安全，且可能需要执行外部防火墙的措施。软件栈中间层的安全问题由用户和运营商共同承担；越低层次的抽象向用户公开，则用户需要越多的责任。Amazon 的 EC2 的用户比 Azure 用户需要承担更多的安全技术责任，而 Azure 的用户又需要比 AppEngine 承担更多的责任。而这些用户的责任可以外包给销售特定的安全服务的第三方公司。像 EC2 这样的平台的一致性和标准化接口使一家公司提供配置管理或防火墙规则分析作为增值服务成为可能。

在云计算可能比较轻易地解决外部面临的安全问题的同时，它又产生了内部安全的新问题。云提供商必须防止由用户造成的盗窃或是拒绝服务攻击。用户需要彼此防范。

目前云计算的主要安全机制是虚拟化。它是一个强力的防御措施，能防止大多数用户之间的相互攻击或是用户攻击云设备。然而，不是所有资源都被虚拟化，且不是所有虚拟化环境都是无漏洞的。虚拟化软件包含着可能使虚拟化代码在某些情况下崩溃的漏洞。不正确的网络虚拟化可能使用户代码访问提供商设施的敏感部分，或者其他用户资源。这些挑战虽然都和管理大型非云的数据中心碰到的挑战相识，不同的应用需要彼此防范。任何的大型因特网服务需要确保一个安全漏洞不会危害其余的全部。

最后一个安全顾虑是防止提供商侵犯用户的权益。提供商肯定控制着软件栈的底层，这有效地绕开大多数已知的安全问题。如果缺少基本的安全技术改进，我们期望用户使用合同和法庭，而不是聪明的安全引擎来制止提供商的不正当行为。一个重要的例外是不小心造成的数据丢失的风险。难以想象 Amazon 监控着虚拟机内存中的内容；可以想象未经格式化就抛弃一个硬盘，或是一个许可漏洞使数据不正确地显示。这是一个非云环境中也存在的问题。标准的防护措施，如用户层加密，在云中也是有效的。这对于云之外的高价值数据已经是非常常见的了，并且已经有可用的工具和专门技术。这种方法已经被 TC3，一家能访问敏感病人的病例和医疗索赔的健康保健公司，在转移他们符合 HIPAA 法案的应用到 AWS 时成功运用。

相似地，审计性可以作为虚拟访客系统接触不到的额外的层加入，提供设备比那些内置和集中软件保密性和设计性相关的责任到单一的逻辑层的应用更加安全。这样提供一个了云计算从特定硬件变到虚拟化能力的新特色。

4.4 数据传输瓶颈

应用不断变得数据密集。如果我们假设应用可能被跨边界的云“撕开”, 这可能使数据的布置和传输复杂化。以 100 150 每 TB 的传输价格，开销会急速飙升，这使得数据传输成本成为一个重要的问题。云用户和云提供商如果想使成本最小化，就必须考虑安置和交通在系统每一级的含义。Amazon 发展新的服务 CloudFont 证明了这种推理。

克服因特网传输的高费用的一个机会是用船运输磁盘。Jim Gray 发现传送大量数

据最便宜的方法就是用船运输磁盘，甚至整台电脑。虽然这不适用于所有情况，但却可以有效处理大量不紧急的点对点传输的情况，例如导入大量数据集合。

为了定量分析这个论据，假设我们想要从 U.C.Berkeley 到华盛顿 Seattle 的 Amazon 用船运输 10TB 的数据。Garfinkel 测量从三个网站到 S3 带宽，并统计平均写带宽为 5Mbits/s 到 18Mbits/s。假设我们有 20Mbits/s 的 WAN 链接。那将花费
10*1012 比特/(20×106 位/秒)=(8×1013)/(2×107) 秒 =4,000,000 秒，
这超过了 45 天。但如果我们通宵用船运输 10 个 1TB 的磁盘，那么将花费少于 1 天的时间来传输 10TB，相当于 1500Mbit/s 的有效带宽。例如，AWS 最近开始提供这样的服务，叫作导入/导出。

4.5 性能不确定性

我们的经验表明多个虚拟机在云计算共享 CPU 和主存的效果出奇地好，但网络和磁盘输入 \ 输出的共享就相当有问题。结果导致不同的 EC2 实例在它们的输入输出的性能与主存的性能相比有很大的不同。我们通过运行 STREAM 内存基准来测量 75 台 EC2 实例。平均带宽为 1,355Mbytes/s，标准差为 52MBytes/s，少于或在平均值的 4% 左右。我们也通过让每台 EC2 实例写 1GB 的文件到本地磁盘来测量 75 台实例的平均磁盘带宽。平均磁盘写带宽在 55Mbytes 每秒左右，标准差为 9MBytes/s 多一点，或者是平均值的 16% 左右。这证明了两台虚拟机之间输入输出干扰的问题。

机会之一是改进架构和操作系统来高效地虚拟化中断和输入输出通道。注意到 IBM 主机和操作系统在 20 世纪 80 年代时就很大程度上克服了这些问题，所以我们有成功的先例可以借鉴。

另一个可能性是闪存能减少输入 \ 输出的干扰。闪存是断电后仍能保存信息的半导体存储器，类似于机械的硬盘，但由于没有移动的部分，它能更快的读写（微秒 vs. 毫秒）和消耗更少的电能。闪存比硬盘每秒每 GB 能承受更多的输出输出存储，所以有冲突输入输出工作流的多台虚拟机能够在同一台电脑中更好的共存，而不会产生机械硬盘带来的干扰。

另一个意料之外的阻碍是担心虚拟机某类批处理程序的计划表，尤其是高性能计算的计划表。考虑到高性能计算 (HPC,High-Performance Computing) 过去常被政府采购。政府花费 $100M 购买 10,000 到 1,000,000 个处理器的超级电脑中心，有很多需要并行处理的任务可以从弹性的计算中获益。今天，许多这些任务在小集群中进行，这通常是低利用率的。取而代之，将这些任务运行在云中的大集群里可能是一个很大的节约。开销关联性意味着使用 20 次，每次使用 1/20 的计算量将不会有惩罚。潜在应用可从中获利，包括那些有很高的潜在经济回报的应用—财务分析、石油勘测、电影特效—甚至愿意花费额外费用换取 20 倍的速度。

有吸引力的高性能计算的障碍不是集群的使用；现在多数并行计算在大的集群中通过信息传输接口 MPI 完成。问题在于很多高性能计算的应用需要确保一个程序的所有进程需要同时运行，且现在的虚拟机和操作系统并不提供程序员可见的方式来确保这一点。因此，克服这个障碍的机会是为云计算提供像“团队计划表”一样的东西。由于之前提到的性能不确定性，所以在云计算环境中实现在传统的团队计划表中相当紧密的同步协调可能是很大的挑战。

4.6 可扩充的存储

先前，我们定义了使云计算具有吸引力的三个属性：短期使用（意味着当需求减少的时候能按比例减少，需求增加时也可以增加），无预付费用，和无限的容量需求。当这些应用到计算中意味着什么已经很清楚了，但如何将它应用在持续的存储上还不是非常清楚。

已经有很多回答此问题的尝试，包括丰富的查询和存储 API，提供性能保证，以及结果一致性的语义学。机会在于创造一个不仅满足现有程序员在持久性、高可用性、管理能力和数据查询方面的期望的存储系统，并且要把它们与云计算能根据需求任意调整的优势相结合。

4.7 大规模分布式系统的漏洞

云计算困难的挑战之一在于去除超大规模分布式系统中的错误。一个通常的情况是这些漏洞不能在小规模的配置中再现，所以必须在生产的数据中心这样的规模下进行调试。

机会之一可能是云计算依赖虚拟机。很多传统的软件服务提供商不使用虚拟机来开发他们的设施，可能是因为他们在虚拟机流行之前就进行了开发，或是他们觉得无法承受虚拟机的性能开销。由于虚拟机是效用计算中所必须的，这种虚拟化层次使获取很多方面有价值的信息成为可能，离开虚拟机是无法完成的。

4.8 快速调整

即付即使用确实适用于存储和网络带宽，两者都按使用的比特量计费。而计算量根据虚拟化的层次，就稍有不同。Google 的 AppEngine 响应负荷的增减，自动调整，并且用户按使用周期付费。AWS 按所占用实例的数量和使用时间收费，即使你的机器是空闲的也需要交费。

机会在于响应负荷并自动地快速按比例增加和减少以节约成本，但不能违反服务

水平协议。确实，加州大学伯克利分校可靠自适应分布系统实验室的关注点之一是普遍和积极地利用统计机器学习的诊断和测量工具来动态调整，自动处理性能和正确性问题，且自动管理该系统的其他方面。

调整的另一个原因是节约资源和成本。由于一台空闲的机器需要使用一台繁忙的机器所消耗电能的 2/3，小心地使用资源能减少数据中心环境的影响，这目前受到很多负面的关注。云计算提供商已经执行了小心且低价的资源消耗计算，效用计算鼓励程序员关注性能（这样，只有在必要的时候释放和获得资源）, 且提供更加直接的操作和开发无效性测量。

意识到成本问题是节约的第一步，但麻烦事容易让人将机器闲置一晚，这样第二天程序员来工作时开机时间是零。一个快速且易于使用的快照/重启工具会更加鼓励节约计算资源。No.9 信誉共享一个用户的坏行为能影响其他使用相同的云的用户的信誉。例如:EC2 的过滤垃圾邮件服务的 IP 地址的黑名单可能限制有效托管的应用。一个机会是发明信誉保护服务，这似于当下提供给托管在正经历这个问题的缩影的小型因特网提供商的服务的”信任邮件”服务（收费）。

另一个法律问题是法律责任的牵连问题—云计算提供商希望他们的顾客承担责任，而不是他们自己（例如发送垃圾邮件的公司将担负责任，而不是 Amazon）。在 2009 年 3 月，FBI 搜捕了 Dallas 的数据中心，原因是一个将服务器托管在此的公司因涉嫌犯罪活动被调查，但一些托管在相同地方的”无辜的旁观者“公司却遭受了几天意料之外的停工，并且有一些甚至停业倒闭。

4.9 软件许可

目前的软件许可通常是限定能运行该软件的电脑。用户购买软件然后支付一年的维护费用。确实，SAP 宣布每年维护费用会至少增加 22% 的软件购买费用，这比较接近 Oracle 的价位。因此，很多云计算提供商起初部分依赖开源软件，因为商用软件的许可模式不合适于效用计算。

主要的机会在于开源保持流行或是商业软件简单地改变它们的许可框架来更好地适应云计算。比如：Microsoft 和 Amazon 现在在 EC2 上为 Windows Server 和 Windows SQL Server 提供即用即付费的软件许可方式。一台运行 Microsoft Windows 的 EC2 实例每小时花费 $0.15，而不是 $0.10 每小时的开源替代软件。IBM 也宣布对托管在 EC2 上的 IBM 软件采用即用即付费的定价政策，价格从 DB2 Express 的 $0.38 每小时到 Lotus 的网页内容管理服务器 IBM WebSphere 的 $6.39 每小时。

5 总结

我们预测了云计算将会增长，所以开发者们应该考虑它。不管一个云服务提供商是以像 EC2 这样的低层次抽象或是像 AppEngine 这样的高层次抽象销售服务，我们都相信计算能力、存储和网络必须都集中注意在虚拟资源的水平伸缩性，而不是单节点的性能。此外：应用软件需要迅速地按比例增加和减少，这是一个新的需求。这样的软件还需要付费使用许可模式来适应云计算。基础设施软件必须意识到它不再运行在裸机上了，而是在虚拟机上。此外，计量计费从一开始就需要内置。硬件系统应被设计成一个容器的规模（至少 10 个台），这是最小的购买量。运行开销将和性能匹配且购买价格的重要性，通过将闲置的内存、磁盘和网络进入低功耗模式奖励能源比例。处理器应该能和虚拟机协同工作，且闪存应该被加入存储层次结构中，且 LAN 交换机和 WAN 路由必须提高带块和功耗。

致谢

Google、Microsoft、Sun Microsystems、Amazon Web Services、Cisco Systems、Cloudera、eBay、Facebook、Fujitsu、Hewlett-Packard、Intel、Network Appliances、SAP、VMWare、Yahoo 支持了一部分的研究。加州工业大学的配套资金/大学合作研究项目（UC Discovery）和国家科学基金给予的支持。

Hadoop 分布式文件系统

摘要：Hadoop 分布式文件系统（HDFS，Hadoop Distributed File System）设计用于可靠存储非常大量的数据集，并将这些数据集以流的形式通过高速带宽发送的用户的应用。在大的集群中，成千台服务器同时直接拥有附加的存储空间和执行用户应用程序的任务。通过分布式存储和多台服务器交叉计算，资源可以根据需求增长，并且对于每种规模的资源需求都很经济划算。本文我们描述 HDFS 的架构和使用 HDFS 来管理 Yahho! 25petabyte 的企业数据感受。

关键词：Hadoop，HDFS，分布式文件系统

1 引言和相关工作

Hadoop 提供了一个分布式文件系统，通过使用 MapReduce 模范分析和转化超大型数据集的框架。Hadoop 一个重要的特点是分割数据且通过多台（上千台）主机进行交叉计算，且并行执行应用数据相关的计算。一个 Hadoop 集群只要简单地增加通常的服务器就能调整计算能力，存储能力和输入输出带宽。Yahoo! 的 Hadoop 集群有 25000 台服务器，且存储着 25 petabyte 的应用程序数据，并且当中最大的集群拥有 3500 台服务器。世界上已经有 100 个其他的组织在使用 Hadoop。

Hadoop 是一个 Apache 的项目；所有的构成都可通过 Apache 开源许可获得。Yahoo! 已经开发和贡献了 Hadoop 核心部分（HDFS 和 MapReduce）的 80%。HBase 起初由 Powerset 开发，现在 Powerset 已经是 Microsoft 的一个部门。Hive 起源和开发于 Facebook。Pig、ZooKeeper 和 Chukwa 起源和开发于 Yahoo!。Avro 起源于 Yahoo! 且和 Cloudera 共同开发。

HDFS 是 Hadoop 的文件系统部分。虽然 HDFS 仿造了 UNIX 的文件系统的接口，但为了提高将来的应用程序的性能而牺牲了标准化接口。

Hadoop 将文件系统的元数据和应用程序数据分开存储。正如其他分布式文件系统，比如 PVFS、Lustre 和 GFS 一样,HDFS 将元数据存储在一台特定的服务器上，称为主节点（NameNode）。应用程序数据存储在其他服务器上，称为数据节点（DataNode）。所有服务器相互连接，并通过基于 TCP 的协议进行通信。

不像 Lusre 和 PVFS，HDFS 的主节点不使用如 RAID 这样的数据保护机制来保持数据的持久性。取而代之，它像 GFS 一样，为了可靠性，将文件的内容被复制到多个主节点上。在确保了数据可靠性的同时，这种策略有提高数据传输带宽的优势，并且有更多的机会来定位所需数据附近的计算资源。

多个分布式系统应经或是正在探索命名空间的真正分布式实现。Ceph 拥有命名空间服务器集群（MDS），且使用动态子树分割算法来平衡地映射命名空间树到 MDS。

GFS 也设计分布式命名空间实现。新的 GFS 拥有成百台命名空间服务器 (宿主)，每台宿主服务器上有 1 亿个文件。Lusre，在 Lusre2.2 版的路标中拥有一个集群命名空间的实现。目地是为了跨多个元数据服务器维护一个目录，每个元数据服务器中包含着命名空间的一部分。一个文件根据文件名通过哈希函数（hash function）被分派给一个特定的 MDS。

2 架构

2.1 主节点

HDFS 命名空间是一种文件和目录的层次结构。文件和目录在主节点上通过 inodes 表示，inodes 记录了诸如许可、修改和访问时间、命名空间和磁盘空间容量之类的属性。文件的内容被分割成大的块（典型的是 128megabyte 每块，but user selectable file-by-file）且文件的各个块独立地复制到多个主节点上（典型的是三个，but user selectable file-by-file）。主节点维护命名空间树和文件块到数据节点（文件数据的物理位置）的映射。一个 HDFS 客户端想要读取第一个包含主节点的文件以获得包含文件的数据块的位置，然后从最靠近客户端的数据节点读取块的内容。当写数据时，客户端请求主节点分配一套三个节点来托管块的副本。然后客户端通过管道将数据写到数据节点。当前的设计是每个集群都有一个主节点。集群中可能拥有成千个数据节点和上万个 HDFS 客户端，这是因为一个数据节点可能同时运行多个应用程序任务。

HDFS 把整个命名空间保存在内存中。inode 数据和构成名字系统的元数据的各个文件的包含块列表统称为映像。保存在本地主机的本地文件系统中的映像的持久记录称为记录点。主节点还保存的映像的修改日志称为在本地主机的本地文件系统中的日记。为了提高持久性，多余的记录点和日记的副本可能由其他服务器产生。在重启过程中，主节点通过读取命名空间和重播日记恢复命名空间。块副本的位置可能随时间而改变，且不作为永久记录点的一部分。

2.2 数据节点

每个数据节点上的块副本由两个在本地主机的本地文件系统中的文件表示。第一个文件包含数据本身，第二个文件是块的元数据，包括块数据的校验和和块的 generation stamp。数据文件的大小等于数据块的实际长度，且不需要额外的空间来 round it up 到像传统文件系统中名义上的块。因此，如果一个块是半满的，它只需在本地分区中占用全满时空间的一半。

在开启过程中，各个节点连接上主节点并进行握手。握手的目的是为了校验命名

空间的 ID 和数据节点的软件版本。如过二者之一和主节点不匹配，则数据节点自动关闭。

命名空间 ID 在格式化的时候就分配给文件系统实例。命名空间 ID 永久存放在集群的所有节点中。不同命名空间 ID 的节点无法将加入这个集群，这样就保证了文件系统的完整性。

软件版本的一致性是很重要的，因为不匹配的版本可能导致数据出错或丢失，而且在数以千记的机器组成的大集群中，很容易忽视那些没有正常关闭的节点的优先级高于软件升级的优先级，或在升级过程中不可用。

刚被初始化且没有任何命名空间 ID 的数据节点，将被允许加入到集群中且获得集群的命名空间 ID。

在握手之后，主节点注册数据节点，数据节点永久保存他们唯一的存储 ID。存储 ID 是一个数据节点的内部标记，就算重启后使用不同的 IP 地址或是端口号，数据节点也能被识别。存储 ID 在数据节点第一次注册到主节点的时候分配，并且不再改变。

一个数据节点通过发送块报告，向主节点确认它所拥有的块副本。一个块报告包含块的 ID，generation stamp 和服务器主机上每个块副本长度。第一个块报告在数据节点注册之后立即发送。后续的块报告每小时发送一次，向主节点提供最新的块副本在集群中的位置信息。

在常规的操作当中，数据节点向主节点发送心跳（heartbeats）来确认数据节点正在运行且它所拥有的块副本可用。默认的心跳间隔是 3 秒。如果 10 分钟内主节点没收到来自数据节点的心跳，则主节点就认为数据节点已经失去服务且数据节点所拥有块副本不可用。之后主节点则计划在其他数据节点上产生新的块副本。

来自一个数据节点的心跳还包含着总存储容量、使用的存储碎片和当前正在传输的数据量的信息。这些主节点用根据这些统计来分配空间和实现负载平衡。

主节点并不直接控制数据节点。它使用答复心跳的方式来向数据节点发送指令。这些指令包括用于：

- 复制块到其他节点
- 删除本地块副本
- 重新注册或关闭节点
- 立即发送块报告

的命令。

这些命令对于维护整个系统的统一性很重要，因此甚至是对大的集群，保持心跳的频率是至关重要的。主节点可以每秒处理成千次心跳，却不影响其他主节点的操作。

2.3 HDFS 客户端

用户应用程序使用 HDFS 客户端访问文件系统，HDFS 客户端是一个输出 HDFS 文件系统接口的代码库。

类似于通常的文件系统，HDFS 支持读、写和删除文件的操作和创建、删除目录的操作。用户在命名空间中通过路径引用文件和目录。用户应用程序通常不需要知道文件系统的元数据和存储分布在不同的服务器上，或是块有多个副本。

当一个应用程序读取一个文件时，HDFS 客户端首先向请求主节点有文件块副本的数据节点列表。然后和一个数据节点直接通信并请求传送想要的块。当客户端写文件时，首先向主节点请求选择一个数据节点来保存文件的第一个块副本。客户端建立一个点到点的管道并发送数据。当第一个数据块充满后，客户端请求新的数据节点来保存下一个块。建立一个新的管道，客户段发送文件的下一部分。每次选择的数据节点可能都是不同的。客户端、主节点和数据节点的交互关系如图 1 所示。

不像通常的文件系统那样，HDFS 提供一个显示文件块位置的 API。这允许像 MapReduce 框架的应用程序能计划将数据放置于何处的任务，这样改进了读取的性能。这也允许应用程序设置文件的复制因子。默认的文件复制因子是 3。对于至关重要的文件或是经常访问的数据，设置更大的复制因子能提高容错性和增加读带宽。

2.4 映像和日记

命名空间映像是文件系统中的元数据，它描述了了应用程序数据的组织形式，是目录还是文件。一个持久印象写在磁盘中的持久记录称为记录点。日记是一个事先写好提交的日志，日记记录了文件系统的永久性修改。对于每个客户端的初始化处理中，修改记录在日记中，且日记文件在提交修改到 HDFS 客户端前会刷新和同步。记录点文件永远不会被主节点改变；当在重启时，管理员或是将在下节中提到记录点节点要求创建一个新的记录点时，将会全部替换覆盖。在启动时，主节点从记录点初始化命名空间印象，然后从日记中重新执行修改直到映像是最新的，文件系统最后的状态。一个新的记录点和空日记在主节点开始服务客户端前被写回到存储目录。

如果记录点或日记丢失或出错，命名空间的部分或全部信息将会丢失。为了保留关键信息，HDFS 可以被设置为在多个存储目录保存记录点和日记。推荐的方法是将目录放在不同的卷上，并且将一个目录放在远端的 NFS 服务器。第一个选择避免一个卷的失效引起数据丢失，第二个选择保护数据免受全部节点失效的影响。如果主节点在将日记写到存储目录之一时发生错误，它将自动把那个目录从存储目录列表中排除。当没有存储目录可用时，主节点将自动关闭。

主节点是多线程系统，并且同时处理来自多个客户端的请求。节约磁盘事务成为了瓶颈，因为所有其他线程需要等待直到由其中一个线程开启的同步刷新和同步的过

程完成。为了优化这个过程，主节点批处理由不同客户端开启的多个事务。当有一个主节点的线程开始刷新和同步操作时，那时所有批处理事务都一起提交。只留下需要检查事务已经被保存且不需要开始刷新和同步操作的进程。

2.5 检查点节点

HDFS 的主节点，除了响应客户端请求的主要角色外，能选择作为其他两个角色之一，作为检查点节点或是备份节点。角色在节点启动的时候指定。

检查点节点每隔一段时间就会组合现有的检查点和日记来产生新的检查点和一个空日记。检查点节点通常运行在不同的主节点主机上，因为它有和主节点一样的内存需求。它从主节点下载当前的检查点和日记文件，本地合并它们，并返回新的检查点给主节点。

创建周期性的检查点是保护文件系统元数据的方法之一。如果所有其他的命名空间映像和日记副本都不可用，那么系统可以从最近的检查点开始。

创建检查点，在新检查点上传到主节点时，让主节点截断日记的尾部。HDFS 集群长时间运行且没有重启，在这段期间内日记会持续增长。如果日记变得非常大，那么丢失数据或日记出错的几率将会增加。并且，一个非常大的日记增加了主节点的重启时间。对于大集群，需要花费一小时来处理一个星期积累下来的日记。创建每日记录点是一个好的方法。

2.6 备份节点

HDFS 最近一个引入的特色是备份节点。像检查点一样，备份节点能够创建周期性的检查点，但除了维护内存中的、最新的系统文件命名空间映像，映像一直和主节点的状态保持同步。

备份节点接受来自活动主节点的命名空间事务的日记流，将它们保存到存储目录，且将这些事务应用到内存中的命名空间映像。主节点把备份节点当作日记存储，就像对待存储目录中的日记文件一样。如果主节点失效，内存中备份节点的映像和磁盘的记录点将记录最新的命名空间状态。

存储节点可以创建新记录点而无需从活动的主节点下载记录点和日记, 因为它在内存中已经有最新的命名空间映像。这使在备份节点上处理记录点更加高效，因为它只需要将命名空间保存到本地存储目录。

备份节点可以看作是一个只读的主节点。它包含了除块位置之外的所有文件系统元数据信息。它可以执行所有普通主节点不涉及修改命名空间和块位置信息的操作。备份节点提供了一种运行主节点却无需永久保存的选择，将命名空间状态持久化的责

任交给备份节点。

2.7 升级，文件系统快照

在软件升级的过程中，系统出错的可能性会因为程序的漏洞或是人为的错误而有所增加。HDFS 中快照的目的是为了在软件升级过程中，最小化对存储在系统中数据的潜在的危害。

快照机制让管理员永久保存文件系统的当前状态，以便如果升级导致数据丢失或出错时，能够回滚升级且让 HDFS 恢复到快照时的命名空间和存储状态。

在系统系统时由集群管理员选择创建快照（只能存在一个）。如果请求一个快照，那么主节点首先会读取检查点和日记，并且在内存中合并它们。然后将新的检查点和空日记写到新的位置，因此旧的检查点和日记保持不变。

在握手的过程中，主节点指导数据节点时候要创建本地快照。数据节点上的本地快照不能通过复制数据文件目录所创建，这样会使集群中每个数据节点的存储容量加倍。取而代之，每个数据节点创建一个存储目录的副本并将存在的块文件硬连接到其中。当数据节点删除一个块时，它只删除硬连接，且添加、修改块时使用写时复制技术。这样旧块副本原封不动保留在旧目录中。

集群管理员可以在重启系统时选择回滚 HDFS 到快照的状态。主节点恢复到建立快照时保存的检查点。数据节点恢复先前重命名的目录和开启一个后台进程来删除快照建立后新建的块副本。选这回滚后，没有前滚的选择。集群管理员可以通过命令系统丢弃快照来恢复快照占用的存储空间，因此结束软件升级。

系统演化可能会导致数据节点的检查点和日记文件，或是块副本文件数据表示的格式发生变化。布局版本识别数据表示格式，且永久保存在主节点和数据节点的存储目录中。在启动每个节点时，比较当前软件的布局版本和保存在存储目录中的本版，且自动将就格式转化成新格式。转化需要在有新软件布局本版的系统重启时，执行快照的新建命令。

HDFS 为主节点和数据节点没有分离布局版本，因为新建快照必须是一个全集群范围的工作，而不是单一节点的事件。如果一个升级的主节点因为软件漏洞删除了它的映像，那么备份唯一的命名空间将仍会导致所有数据丢失，因为主节点将无法识别数据节点发送的块报告，且将命令删除它们。这种情况下回滚将恢复元数据，但数据本身将丢失。这时就需要一个候选的快照来防止大破坏。

3 文件输入输出操作和副本管理

3.1 读写文件

一个应用程序通过新建文件和将数据写到文件中的方式在 HDFS 上添加数据。在文件关闭之后，已经写入数据不能被更改或删除，除非重新打开这个文件添加新数据。HDFS 实现了单一写者，多个读者的模式。

打开并写文件的 HDFS 客户端被授予该文件的一个租约；其他客户端不能写这个文件。执行写操作的客户端通过向主节点发送心跳来更新租期。当文件关闭时，租期被收回。

租期由软限制和硬限制共同约定。在软限制过期之前，写者将独占文件的访问权限。如果软限制过期了且客户端没能关闭文件或更新租期，另一个客户端就可以占用租期。如果在硬限制（1 小时）过期之后，客户端仍然无法更新租期，HDFS 就假设这个客户端已经退出并自动代替写者关闭文件，并恢复租期。读者的租期并不阻止其他客户端读取文件；一个文件可以同时被多个客户端读取。

一个 HDFS 文件包含很多块。当需要创建一个新块时，主节点分配一个唯一的块 ID 给新块，并决定哪一个数据节点的列表来保存块副本。数据节点建立管道 the order of which 最小化整个网络从客户端到最后一个数据节点的距离。比特流作为包序列被压进管道中。应用程序在客户端上将比特流写入第一个缓存。当一个包的缓存填满之后（通常是 64KB），数据就被压入管道中。可以在收到前一个包的确认之前将下一个包压入管道。outstanding 包的数量由客户端的 outstanding 包窗口大小限制。

在数据写入一个 HDFS 文件后，在文件关闭之前，HDFS 并不保证数据对新的读者可见。如果一个用户应用程序需要可见保证，它可以显式调用刷新（hflush）操作。然后当前的包将立即被压入管道，并且刷新操作会一直等待直到所有管道中所有数据节点确认成功传输包。所有在刷新操作之前写入入的数据将对读者可见。

如果没有错误发生，图 2 表示了一个有三个数据节点的管道和一个有 5 个包的块，块的构造经过如图 2 所示的三个阶段。图中，粗线表示数据包，虚线表示确认信息，细线表示控制建立和关闭管道的信息。垂直线表示客户端上的活动和三个数据节点，时间从上到下增加。从 t0 到 t1 是管道的建立阶段。t1 到 t2 是数据流阶段，t1 是第一个数据包发送的时间，t2 是收到最后一个包的确认的时间。这有一个刷新操作传输第二个包。刷新指令和数据包一起传输，而不是分开的操作。最后 t2 到 t3 是这个块的管道关闭阶段。

在一个有上千个节点的集群中，节点失效（最常见的存储错误）每天都在发生。一个存储在数据节点的副本将因为内存、磁盘或网络的错误而出错。HDFS 为 HDFS 文件的每个数据块产生和存储校验和。校验和由 HDFS 客户端在读取时校验，以帮助检

查由客户端、数据节点或是网络引起的错误。当客户端新建一个 HDFS 文件时，它计算每个块的校验和序列并将数据发送给数据节点。数据节点在元数据文件中存储校验和，和块数据文件分开存储。当 HDFS 读取一个文件，每个块数据和校验和将传送给客户端。客户端计算为接受到的数据计算校验和并检验新计算出的校验和是否和它接受到校验和相匹配。如果不匹配，客户端通知主节点副本出错，然后从从另外的数据节点获取一个不同的块副本。

当一个客户端打开一个文件进行读取时，它从主节点获取块列表和每个块副本的位置。每个块的位置按它们离读者的距离排序。当读取一个块的内容时，客户端首先尝试最近的副本。如果读取尝试失败，客户端尝试序列中的下一个副本。目标数据节点不可用，或是在检查校验和时发现副本出错，可能导致读取失败。

HDFS 允许客户端读取正在写入的文件。当读取一个正在写入的文件时，最后正在写入的块的长度对于数据节点是不可知的。这种情况下，在开始读取内容之前，客户向一个副本端请求最新的长度。

HDFS 的输入输出是特别为像 MapReduce 这样需要大量顺序读写的批处理系统设计的。然而，为了支持像 Scribe 这样提供实时数据流给 HDFS，或是像 HBase 这样提供随机的实时表访问的应用程序，在改进它的读写响应时间上做了很多努力。

3.2 块放置

对于一个大的集群，flat topology 连接上所有节点是不现实的。一个通常的方法是将节点分散在多个机架上。一个机架上的节点共享一台交换机，且机架交换机由一台或多台核心交换机连接。不同机架上的两个节点的通信必须通过多台交换机。在大多数情况下，同机架节点之间的网络带宽大于不同机架节点之间的网络带宽。图 3 描述了有两个机架的集群，每机架包含 3 个节点。

HDFS 通过两个节点之间的距离估计它们之间的网络带宽。假设从一个节点到它的父节点的距离是 1。通过把把两个节点到他们最近的共同祖先的距离相加可以计算两个节点之间的距离。两个节点之间的距离越短，意味着它们可用于传输数据的带宽越大。

HDFS 允许管理员设置脚本来以一个节点的地址作为输入，返回节点的机架信息。主节点是解决各个数据节点机架位置的中心。当一个数据节点注册到主节点时，主节点运行一个配置脚本来决定该节点属于哪个机架。如果没有配置脚本，主节点就假设所有节点都属于一个默认的机架。

副本的放置是 HDFS 数据稳定性和读写性能的关键。一个好的副本放置策略能提高数据稳定性、可用性和网络带宽利用率。目前 HDFS 提供可配置的块放置策略接口，这样用户和研究员就可以实验和测试哪一个策略对于他们的应用程序是最优的。

默认的 HDFS 块放置策略提供一个最小化写代价和最大化数据稳定性、可用性和总的读带宽之间的 tradeoff。当新建一个新块时，HDFS 将第一个副本放置在读者所在的节点，第二和第三副本放置在两个不同机架上的不同节点上，剩下的放置在随机的节点上，但是要求在副本的数量少于机架的两倍时，一个节点上放置不多于一个副本，同一机架上放置不多于二个副本。放置第二和第三个副本在不同机架上的选择更好地将一个文件的块副本分配到整个集群中。如果头两个副本被放置在同一机架上，那么对于任何文件，它 2/3 的块副本都将放置在同个机架上。

在选择所有目标节点后，节点按照它们到第一个副本从近到远的顺序被组织成管道。数据按这样的顺序被压入管道。对于读取，主节点首先检查客户端主机是否位于这个集群中。如果是，那么将块位置按它到客户端的距离由近到远的顺序返回给客户端。从数据节点中按照优先顺序度读取块。（MapReduce 应用程序通常运行在集群节点上，但只要主机能连接上主节点和数据节点，那么它就能运行 HDFS 客户端。）

这种策略减少中间机架和中间节点写传输量，且通常能提高写性能。因为机架失效的可能性远远低于一个节点失效的可能性，这中策略没有影响数据的可靠性和可用性保证。在三个副本的通常情况下，在读数据时能减少使用的总网络带宽，因为一个块只放置在两个不同的机架而不是三个。

默认的 HDFS 副本放置策略可以总结如下：

- 没有数据节点包含多余一个任意块的副本。
- 没有机架包含多余两个相同块的副本，在集群中提供足够的机架。

3.3 副本管理

主节点努力确保每个块总是拥有一定数量的副本。当接受到一个来自数据节点的块报告时，主节点检测一个块副本数是低于或高于标准数量。当一个块副本数过多时，主节点选择一个副本进行删除。主节点首先倾向于不减少持有副本的机架的数量，其次倾向于从可用空间最少的数据节点中删除副本。目的在于平衡数据节点之间的存储空间利用率，同时不降低块的可用性。

当一个块副本数量变得过少时，它就被放入复制优先队列。只有一个副本的块拥有最高优先级，同时副本数量大于它复制因子的 2/3 的块优先级最低。一个后台线程周期性地扫描复制队列的队头来决定把新副本放在哪。块的复制遵循相识于新块放置的策略。如果现有的副本数为 1，HDFS 将下一个副本放置在不同的机架上。在块有 2 个副本的情况下，如果连个副本在同一机架上，那么第三个副本就放在一个不同机架上；否则，第三个副本就放在现有副本的机架的一个不同节点上。目的是为了减少创建新副本的代价。

主节点还确保并非所有块副本都位于一个机架上。如果主节点发现一个块的副本

都在同一机架上，那主节点就把该块当作副本数过少，同时使用前面提到的相同的块放置策略复制块到一个不同的机架上。在一个主节点接受到新副本已经创建的通知后，块变为副本过多的状态。之后主节点将决定删除一个旧副本，因为过多复制策略倾向于减少机架的数目。

3.4 平衡器

HDFS 块放置策略并不计算数据节点的磁盘空间利用率。这是为了避免新数据（更可能是引用的数据）成为数据节点的一个子集。因此在数据节点中数据可能不总是一致地放置。当新结点加入集群时也会发生不平衡。

平衡器是一个平衡 HDFS 集群上的磁盘使用的工具。把一个阈值作为输入参数，阈值的范围是 0 到 1 之间。如果每个数据节点，节点的使用率（节点上使用空间占总空间的比率）和整个集群的使用率（集群中使用空间占总空间的比率）的差值不超过阈值，那么一个集群就达到平衡。

这个工具被设置成一个可以被集群管理员运行的应用程序。它迭代地将副本从高利用率的数据节点移动到低利用率的节点。平衡器的一个关键要求是维护数据的可用性。当选择一个副本进行移动和决定它的目标位置时，平衡器保证这个决定不会减少副本或是机架的数量。

平衡器通过最小化中间机架数据拷贝来优化平衡进程。如果平衡器决定一个副本 A 需要移动到不同的机架，且目标位置恰好有相同块的副本 B，那么数据将从副本 B 除拷贝而不是副本 A 处。

第二个配置参数限制了再平衡操作消耗的带宽。所允许的带宽越大，集群达到平衡状态的速度越快，但是和应用程序进程竞争得越厉害。

3.5 块扫描器

每个数据节点运行一个块扫描器周期性地扫描它的块副本且检验存储校验和是否和块数据匹配。在每个扫描过程中，块扫描器为了在配置的时间内完成检验而调整读带宽。如果客户端读取一个完整的块且校验和验证成功，它将通知数据节点。数据节点就把它作为副本的检验。

每个块的校验时间存放在可读日志文件。在任何时间，最多只有两个文件在最高层的数据节点目录中，目前和之前的日志。新的检验情况将追加到目前文件中。每个数据节点相应地有一个在内存中按副本检验时间排序的扫描列表。

任何时候一个读客户端或是块扫描器检测到一个错误块，它将通知主节点。主节点标记副本出错，但并不立即删除副本。而是开始复制一个好的块副本。只有当好的副

本数量达到了块复制因子时，才将出错副本删除。这个策略是为了尽可能久地保存数据。所以即使所有块副本都出错，这个策略能让用户从出错副本中取回数据。

3.6 退役

集群管理员通过列出允许注册的节点的主机地址和不允许注册的节点的主机地址来指定哪些节点可以加入集群。管理员可以命令系统重新评估这些包含和排除列表。一个集群现有成员变成排除状态时，就被标记为退役。只要一个数据节点标记为退役，它就不会作为副本目标位置被选中，但它将继续服务读请求。主节点开始计划复制块到其他数据节点。只要主节点发现退役数据节点所有块被复制，那么数据节点就进入退役状态。之后它就可以安全地从集群中移除而不危害任何数据的可用性。

3.7 集群之间数据拷贝

当处理大数据集时，复制数据近出 HDFS 集群是令人畏惧的。HDFS 提供了一个叫作 DistCp 的工具用于大数据在集群间或集群内的并行拷贝。它是一个 MapReduce 任务。每个 map 任务复制源数据的一部分到文件系统中的目标位置。MapReduce 框架自动处理并行任务计划、错误检测和恢复。

4 在 Yahoo! 的实践

在雅虎的大型 HDFS 集群包括大约 3500 个节点。一个典型的集群节点的配置如下：

- 双核 Xeon 处理器, 频率 2.5ghz
- 红帽企业版 Linux 服务器 5.1 版
- Sun Java JDK 1.6.0 13-b03
- 4 个直接连接的 SATA 硬盘（每个 1terabyte）
- 16G 内存
- 1GB 以太网

70% 的磁盘空间分配给 HDFS。剩余的空间留给操作系统（红帽 Linux）、日志和存储 map 任务的输出流。（MapReduce 的中间数据并不存放在 HDFS。）40 个节点在同一个机架上共享一个 IP 交换机。机架交换机和 8 台核心交换机台台相连。核心交换机提供机架和集群外资源连接。每个集群，主节点和备份节点主机特别配备了最多 64GB 的内存；应用程序任务永远不会分配给这些主机。总体上，一个 3500 个节点的集群有 9.8PB 的存储空间可用，相当于 3.3PB 应用程序存储空间的块复制 3 次。作为

一个方便的近似，一千台节点代表 1PB 的应用程序存储。在 HDFS 被使用的这些年（和将来），作为集群节点的主机将从改进的技术中获益。新集群节点总是拥有更快的处理器，更大的磁盘空间和更大的内存。较慢、较小的节点将从集群中回收或撤离，然后用于 Hadoop 的研发和测试。如何准备数据节点的选择很大程度上是一个经济采购计算能力和存储空间的问题。HDFS 不强制使用一个特定的计算能力存储空间比率，或是给连接到集群的节点设定一个存储容量的限制。

举一个大型集群（3500 个节点）的例子，有近乎 6 千万个文件。这些文件有 6.3 千万个块。每个块通常复制三次，每个数据节点有 54000 个块副本。集群中每天用户应用程序会新建 2 百万个新文件。在 Yahoo! 的 Hadoop 集群中，25000 个节点提供了 25PB 的在线数据存储。从 2010 年开始，这是 Yahoo! 一个中型但不断发展的数据处理框架的部分。2004 年，Yahoo! 开始调查有分布式文件系统的 MapReduce 编程。2006 年，Apache 的 Hadoop 项目成立。2006 年底，Yahoo! 已经将 Hadoop 用于内部使用并且拥有 300 个节点的集群用于开发。从此 HDFS 成为 Yahoo! 后台管理系统不可分割的一部分。HDFS 标志性的应用程序是网站地图的产生，这是一个万维网的索引，是搜索至关重要的部分（75 小时的时间，产生了 500terabyte 的 MapReduce 中间数据，300terabyte 的总输出）。更多的应用程序迁移到了 Hadoop 上，尤其是那些分析用户行为并建模的程序。

成为 Yahoo! 的技术套件的关键组成，意味着在解决研究项目和管理很多 petabyte 的企业数据的问题之间存在差异。更重要的是稳健性和数据持久性的问题。但是经济的性能、用户社区成员之间共享资源的准备和系统操作者的简便管理也很重要。

4.1 数据持久性

数据复制三次是一个防止因不相关节点失效引起数据丢失的健壮性措施。采用这种方法，Yahoo! 从没有丢失过一个块。对于大型集群，一年之中丢失块的可能性小于 0.005。关键在于能理解节点每个月有 0.8% 的几率失效。（即使节点最终被恢复了，但却没办法恢复它之前保存的数据。）所以对之前描述的大型集群样本，每天会丢失 1 或 2 个节点。相同的集群会在 2 分钟左右重新新建 54000 个失效节点上的块副本。（重新复制很快，因为这是一个随着集群规模而伸缩的并行问题。）几个节点在两分钟之内失效而导致一些块副本丢失的可能性相当小。

相关节点的失效则是一个不同的威胁。在这方面，最常见出现的错误是机架或核心交换机的失效。HDFS 可以容许丢失一台机架交换机（每个块还有副本在其他的机架上）。一台核心交换机的失效可能有效地断开一片集群和多个机架的连接，这种情况下一些块将会不可用。在两种情况下，修复交换机可将不可用的副本恢复到集群中。另一种相关失效是集群意外或有意断电。如果断电跨越了机架，很有可能一些块将变得不

可用。但是恢复供电可能不能修复丢失数据，因为有 1.5% 的节点不能在全加电重启中存活。跟据统计和实践，一个大集群将在加点重启过程中丢失一小撮块。（在几星期内，用每次有意重启一个节点的策略来鉴别不能在重启中存活且还没被检测到的节点。）

除此之外全部节点失效，那么存储的数据可能出错或丢失。块扫描器每两星期扫描以此大集群中的所有块，并且在这个过程中发现约 20 个坏的副本。

4.2 照顾下议院

随着 HDFS 的使用日益增长，文件系统本身必须引入在一个大型且不同的用户社区中共享资源的方法。第一个这样的特点是权限框架，它模仿了 Unix 文件和目录的权限管理。在这个框架中，文件和目录对于所有者、文件或目录所有者所在群体的其他成员和所有其他用户有不同的访问权限。Unix（POSIX）和 HDFS 之间的根本不同在于在 HDFS 上普通文件只拥有“可执行”权限或“粘着”位。

在目前的框架中，用户标识弱：主机说你是谁就是谁。当访问 HDFS 时，应用程序客户端简单地在本地操作系统查询用户标识和群体成员。一个更强标识的模型正处于研发之中。在新的框架中，应用程序客户端必须向名字系统出示从可信任源获取的凭据。不同凭据的管理是可能的；最初实现将使用 Kerberos。用户应用程序可以使用相同的框架来确认名字系统也有可信任标识。并且名字系统还可以要求从集群中的每个数据节点获取凭据。

总的可用数据存储空间由数据节点的数量和为每个节点配备的空间所确定。HDFS 的早期实验中证明需要一些方法来强制执行资源块用户社区分配的策略。不仅需要强制执行公平共享，在用户应用程序可能涉及在上千台主机上写数据时，避免应用程序偶然耗尽资源也很重要。对于 HDFS，因为系统元数据永远在内存中，命名空间的大小（文件和目录的数量）也是一项有尽的资源。为了管理存储空间和命名空间资源，可能给每个目录分配一个限额，限额也由子树中总的文件和目录数决定。

虽然 HDFS 的架构假设大多数应用程序会将大量数数据流作为输入，但是 MapReduce 程序框架倾向于产生很多小的输出文件（每个 reduce 任务产生一个），给命名空间资源造成更大的压力。方便起见，目录子树压缩成一个 Hadoop 压缩文件。一个 HAR 文件类似于 tar、JAR 或是 Zip 文件，但是文件系统的操作可以把个别的文件打包压缩，且 HAR 文件可以透明地用作 MapReduce 任务的输入。

4.3 基准程序

HDFS 的设计目标之一是为大数据集提供很高的输入输出带宽。有三种测量标准来检测这个目标。

- 从人造基准观测到的带宽是多少？
- 在一个最少用户任务的生产集群中观测到的带宽是多少？
- 一个最精心构造的大型用户应用程序能获得得的带宽是多少？

这里的统计报告是从至少 3500 个节点的集群中获取的。在这种规模下，总带宽和节点数量线性相关，所以我们感兴趣的统计数据是每个节点的带宽。这个基准程序作为 Hadoop codebase 的一部分是可用的。

DFSIO 基准程序衡量了读、写和追加操作的平均数量。DFSIO 是一个可用的应用程序，它作为 Hadoop 分布式系统的一部分。MapReduce 程序从大文件中读/写/追加随机的数据，或是将随机数据读/写/追加到大文件中。这个任务中的每个 Map 任务在不同的文件上执行相同的操作，传输相同数量的数据，并且将它的传输速率报告给单个 reduce 任务。然后 reduce 任务汇总这些测量。这个测试不需要争用其他应用程序资源就可以运行，并且会选择和集群规模相称的 map 任务数量。它设计只是用来测量数据传输时的性能，并且排除任务计划、启动和 reduce 任务造成的开销。

- DFSIO 读速度：66MB/s 每节点
- DFSIO 写速度：40MB/s 每节点

对于一个生产集群，读取和写入的比特数将报告给一个 metrics 采集系统。这些平均值将被接管几个星期，且通过上百个独立独立用户表示集群的利用率。平均来看，任何时刻 1 或 2 个应用程序任务占用 1 个节点（少于可用的处理器核数）。

- 繁忙集群的读速度：1.02MB/s 每节点
- 繁忙集群的写速度：1.09MB/s 每节点

表 2

在 2009 年年初，Yahoo! 参加了 Gray Sort 竞赛。这个任务的目的是测试系统移动文件的能力所能承受的压力（和排序没有关联）。竞赛方面意味着表 2 的结果是在现有设计和硬件条件下，最好的用户应用程序所能达到的结果。最后一列的输入输出速度是从 HDFS 读取输入和将输出写入 HDFS 的总和。在第二行，虽然 HDFS 的速率减少，但每个节点总的输入输出将近翻了一倍，因为对于大数据集（betabyte 规模），MapReduce 中间结果也必须写入磁盘和从磁盘中读取。在较小规模的测试中，不需要将 MapReduce 的中间结果写入磁盘；它们缓存在这个任务的内存中。

大型的集群需要 HDFS 主节点支持大集群所期望达到的客户端操作数。NNThroughput 基准程序是一个单一节点的进程，它开启主节点应用程序且相同节点上运行一些列客户端线程。每个客户端线程通过直接调用主节点实现这些操作的方法重复执行相同的主节点操作。基准程序测量主节点每秒执行的操作数。这个基准程序设计上避免了 RPC 连接和序列化造成的通信的外开销，因此在本地运行客户端而不是从不同节点上远程运行。这里表 3 提供了主节点净性能的上限。

5 后续工作

这部分展示一些 Yahoo 的 Hadoop 团队正在考虑的一些后续工作；Hadoop 是一个开源项目意味着新的功能和改变大部分由 Hadoop 开发社区决定。

当主节点失效时，将导致 Hadoop 集群不可用。考虑到 Hadoop 主要用于批处理系统，重启主节点已经是一个令人满意的方法。然而，我们已经像自动的失效备份发展。目前的备份节点接受所有来自主要的主节点的事务。这将允许将失效备份转移到一个可用的或甚至是繁忙的备份节点上，如果我们发送块报告到主要的主节点和备份节点的话。除 Yahoo! 之外的一些 Hadoop 用户已经尝试手动失效备份的实验。我们的计划是使用 Zookeeper，Yahoo 的分布式一致技术来建立一个自动的失效备份方案。

主节点的伸缩性已经成为一个关键。因为主节点存有所有的内从中的命名空间和块位置，主节点堆的大小已经限制了文件的数量和块地址的数量。主节点主要面临的挑战是当内存的使用接近上限时，主节点会因为 Java 的垃圾回收机制而变得响应迟钝，并且有时需要重启。虽然我们鼓励我们的用户创建大文件，但这还没发生，因为它要求改变应用程序的行为。我们已经添加了一些配额来管理使用情况，且提统一个压缩工具。然而这些没有根本上解决伸缩性的问题。

我们近期的伸缩性解决方案是允许多个命名空间（和主节点）在集群内共享物理存储空间。我们通过块池（block pool）标识符扩展块 ID 作为前缀。块池和 SAN 存储系统的 LUNS 相识，且有块池命名空间的大小和文件系统容量相识。

这个方法相当简单，而且需要对系统的改变最少。除伸缩性之外，它还提供了一些其他优势：它分离了不同应用程序集的命名空间且提高了集群的整体可用性。它还产生了块存储的抽象，让其他服务能用不同的命名空间架构来使用块存储服务。我们计划去探索其他伸缩性的方法，例如只将部分命名空间保存在内存中和将来主节点真正的分布实现。特别地，应用程序会产生少数的大文件的假设是有瑕疵的。如之前所提到的，改变应用程序行为是很困难的。此外，我们见过新型的 HDFS 应用程序需要存储大量的小文件。

多个独立命名空间的不足之处在于管理成本，特别是如果命名空间的数量特别多时。我们也计划使用应用程序或以任务为中心的命名空间而不是集群为中心的命名空间—这类似于 80 年代末 90 年代初，在分布式系统中用于处理远程执行的按照进程的命名空间。

目前我们的集群少于 4000 个节点。我们相信我们可以通过上述解决方案扩充成更大的集群。然而，我们相信多个集群将更加的明智合理，而不是单一的大集群（也就是说 3 个 6000 个节点的集群而不是 1 个 18000 个节点的集群），因为它允许提高了可用性和独立性。为了达到这个最终目标，我们计划在集群之间提供更多的协作。比如缓存远程访问文件或是当文件集跨集群复制时，减少块的复制因子。

致谢

致谢 我们想要感谢现在或之前 Yahoo! HDFS 团队的所有成员为建立这个文件系统所作出的努力。我们想要感谢所有的 Hadoop 委员会成员和合作者的宝贵贡献。Corinne Chandel 为这篇文章绘制了图表。

基于 ERP 原则的生产管理信息系统

摘要：ERP（Enterprise Resources Planning , 企业资源计划）作为一种先进的管理思想，在国外已经有很多成功实施的案例，并且在我国，在实施方面已经做了很多的探索；在现阶段，ERP 在国内企业的有效实现和实施是一个需要深入研究的问题。根据国内航空公司的运营模式，并经过调查和分析生产管理在其中的地位和作用，我们提出了建立一套先进的、符合企业实际情况的生产管理系统的目标，引进系统设计，澄清系统在功能和架构上的细节，并且探讨设计方法和其中的关键技术。在系统设计上使用快速原型的方法，并改进快速原型方法的流程。在相同应用上，本文的系统设计思想有一定的参考价值。**关键词：**ERP，信息化，原型设计

1 引言

ERP（Enterprise Resources Planning, 企业资源计划）作为一种先进的管理思想、方法和实现，它的实施成果和企业的运营管理模式有直接的关系。通过对我国航空引擎生产企业的管理模式、产品结构和生产的调查和分析，虽作为复杂产品结构，但在工厂和产品管理、管理模式和生产流程的特殊问题上，和 ERP 系统相比还存在一定的差距。从生产管理的角度出发，本文为 ERP 的实施提出了一系列的指导。其中不但包含了 ERP 原则的本质，还有助于了 ERP 在企业中的成功实施。我们按照企业综合计划的观点，集中于生产管理方面，引用 ERP 的开发思想，自主研发了一套生产管理系统。它的成功实施表明了国内企业在信息技术的独立研发上已经成熟，取代了先引进再进行二次开发的方法。同时，本文在此探讨在同行的国有企业中的信息化问题。

2 系统设计

作为企业管理的关键要素之一，生产管理系统对于实现企业的有效管理和实现目标是非常重要的。ERP 作为企业管理的核心，同时也是 ERP 原则的核心部分。ERP 的逐步实施会取得有利的优势，并以此为突破点。基于这点，我们已经取得了富有成效的研究和发展。

2.1 系统设计目标

生产管理系统的设计目标是建立一套先进的、现代化的生产管理信息系统。这套系统能够反映出 ERP 原则，并且满足国内企业的实际需求，完成年度计划、需求计划、月份计划、每周计划、季度计划、计划监控和计划评估的信息化管理。

2.2 设计分析

在准备阶段，我们调查了企业运营的流程，组织架构，技术力量和现有情况等等。我们进行了详细的调查。企业有如下情况：

- 整体上，重事电脑管理的基层和专业人员具备了电脑管理系统开发的思想；
- 目前部分管理已经采用了电脑辅助管理，并且电脑技术在生产管理中扮演了重要的角色；
- 明白生产管理信息的自动化和网络处理需求是非常紧急的；
- 在企业里，网络层次结构的建设能够满足日后管理系统的需求，为生产管理系统的实施准备了良好的环境和条件。经过反复的交流和讨论，我们确定了明确的需求。

基于调查和分析的最终结果，我们对企业的运营流程进行了分类。根据这个运营的流程，我们确立了工作流。我们把所有流程中的关键数据进行抽象和分类，以建立一张标准数据表。这张表就是该系统数据库的原型，它包含了数据库和表的架构。经过我们的分析和总结，对原型设计进行了几次改进，最终确立了操作信息处理的流程图（如图 1 所示）的原型。

在设计过程中，我们着重讨论和分析了各个模块数据之间的关系。当前管理的流信息的逻辑关系难以满足 ERP 的整体思想。各个模块数据之间的关系已经比较精确了，但还不够正确；各个独立模块的材料之间没有联系，但所有数据整体上却是相互联系的。我们避免了 ERP 组织良好的一致性，且采用了部分严密而整体灵活的解决方案以减少刚开始实施时的难度。严密一致得像锁链一样的问题将在后续开发严密一致的产品目录、年度计划概要和和分散的日历表的阶段中给予考虑和解决。日历表和每月计划是紧密相关联的，然而工厂中的存货和在制品的关系整体上却是灵活的。这不但是该系统的独到之处，也是其成功实施的关键所在。

2.3 系统架构

根据实际需求，为实施而特制的设计思想在本系统的架构之中体现得淋漓尽致。所有的模块都按照用户的实际操作流程划分，并强调软件的实用性。在操作的设计上，该系统采用了简洁的菜单，完全符合一般的操作标准和使用。系统包含这些模块：年度计划、需求计划、每月计划、每周计划、季度计划、计划监控和计划评估、数据维护等等，如图 2 所示

3 系统设计方法

由于本次系统设计目标和国外成熟 ERP 系统在实现上的巨大差异，在系统的研究和开发阶段非常有必要处理和设计一套关于系统的架构关系、功能布局、算法思想和技术措施的方案。该系统的设计方法主要包含以下方面。

- 这是一个生产管理系统，但客户所有的功能需求都类似于 ERP。该系统有更大的规模，尽管客户有外在的需求，但由于研发企业管理系统的难度，似乎软件的开发和实施需要很长的路要走。用户要求该系统和现在的管理系统相同，增加了软件开发的不确定性，所以很难以一个简单的方法完成研究和开发。为获得成功，本系统采用了软件工程中的快速原型模式。在实施的过程中，我们做出了一些改进，比如：在获得对每个测试原型的确认之前，对模块和算法采取了简化。考虑到多方面因素，比如软件的实现和操作功能、功能的效率、时间复杂度等等。我们尽可能地和用户沟通。我们为原型的开发选择了可行的模型，且避免了弃件式的方法，所以模块是可重用的。这不仅使需求变得清晰，也使得原型更加接近目标，功能更加适应实际情况。
- 在设计系统软件的阶段，我们充分利用了快速原型。在几次原型设计之后，各个原型和算法之间的关系都经过测试和改进，这有助于在短时间内确定系统的架构。这增加了用户的信心并且保证了软件开发的成功。
- 模块测试：由于算法的复杂性和可行性，模块的算法会通过操作和收集的样本立即进行测试，并且检查出在这阶段程序中可能存在的问题，这些问题将在系统测试和点对点测试阶段分别进行解决。模块测试为大量数据的导入做好了准备。
- 数据导入测试和安排：大量的生产列表和处理信息等等将在这阶段被导入。通过对比导入操作的结果和手工操作的结果，对算法和可能存在的问题进行测试，尤其是对数据有效性和合理性的评估，数据的校正和排放。
- 系统测试：在这阶段，完成主控链接系统的测试，以解决算法和数据中可能存在的问题，并验证之前想法的可行性。
- 导入并测试之前的业务信息：在本阶段，相对完整的先前的业务信息（主要是物品信息）将被导入。完整的需求信息将作为输入，并且将在分析操作结果后检验它的完整性和有效性，以便进一步改进算法。
- 系统操作分析：通过对比系统化的计划和手工的计划，检查和分析它们的不同之处来发现这阶段存在的问题，这样来对算法或数据进行修正和改进。
- 系统功能评估：通过对比和分析系统功能的操作和测试结果，以及评估系统功能的实现，然后讨论系统的可行性和实现目标，且提出改进的需求。

4 关键技术和问题

为了解决在生产管理实践中碰到的问题，系统设计和应用了一致的处理方法和技术。

4.1 数据维护

产品层次树的算法被设计成能够适应多种结构，且数据结构被设计成能够适应安排的快速分解算法。

4.2 计划管理

通过对网络计划技术的前进和后退计算方法的充分利用，我们设计了一种支持分解的安排模板；使用分解的模板，为需求计划和净组合需求计划设计了快速算法。特别是组合算法上考虑到在交货时间上的多种系数。作为系统的特色之一，它计算了在网络计划中最早的和最近的时间，且关键节点完全清楚地知道不同的时间；比较容量需求和实际的容量来调整主计划安排，使之和 ERP 一样准时，且考虑到目前的生产管理情况；工厂中每月计划的产生和完成的评估算法考虑到了整体上大部分管理和计划的准确性，这有助于解决由于干扰信息链（来自生产计划和工厂开始计划的干扰信息，来自计划和进出物品的干扰信息，批量管理的限制等等）产生的毫无联系的信息。

5 总结

现在在国家不断提倡发展信息化尽设的环境下, 国内很多企业在不同地层次上着手开始信息化建设的工作，使企业管理的效率和层次更上一层楼，提高企业的市场竞争力。在本文中提到的调研和实施的团队，双方都紧紧抓住了当前的机会，相互合作，共同努力克服困难，用尽全力找出问题的关键和突破口，并基于对企业管理流程的本质的深刻和详细分析来解决问题，最终研发出一套适合企业实际需求的、基于 ERP 原则的生产管理系统。成功实现对 ERP 原则的实施。后来，随着 ERP 实施的深入和综合，非常有必要进行规模庞大且细致的工作；企业的管理模式难免需要适当的调整。这是一个循序渐进的、漫长的过程，还需要长期的努力。