蓝鲸计费系统技术白皮书

Version 2.8

版权所有©：2005-2022，上海重玄科技，保留所有权利

版本控制

版本号	修改时间	修改内容	修改人	审阅者
1.0	2007-12-08	创建	白杨	赵东
1.1	2007-12-10	根据审阅者意见修改	白杨	
2.0	2013-01-16	更新至蓝鲸计费 V2 版	白杨	
2.1	2013-01-17	调整章节顺序；修正少量笔误	白杨	
2.2	2013-08-09	更新远程单元相关插图	白杨	
2.3	2013-11-27	修正一处笔误	白杨	
2.4	2016-08-08	新增 5. 技术特点小节	白杨	
2.5	2016-12-10	新增 5.2.3.3 基于 BYPSS 的高性能集群小节	白杨	
2.6	2017-07-06	更新 3.2.2 CDR 服务器设计框架中，对 HAC+HPC 集群的描述	白杨	
2.7	2020-08-18	更新全球专利证书一瞥	白杨	
2.8	2022-02-20	对 BYPSS、BYDMQ 以及 BYST 等组件的个别细节进行澄清	白杨	

目录

版本控制......I

目录......II

1. 概述......1
1.1 行业背景......1
1.2 系统描述......1
1.3 技术指标......10
1.4 应用支撑平台......11
1.4.1 单点支撑千万量级并发的高效 IO 服务器组件......12
1.4.2 强一致多活 IDC 高可用（HAC）和高性能（HPC）服务器集群......12
1.4.3 高效、高强度的密码编码学组件......19
1.4.4 数据查询分析引擎......19
1.4.5 更多......20

2. 系统总体架构......1
2.1 数据库......2
2.2 PBX 网络......2
2.3 CDR 服务器......2
2.4 Web 客户端......2

3. 系统总体设计......4
3.1 应用支撑平台......4
3.1.1 应用支撑平台功能描述......5
3.1.2 应用支撑平台设计框架......6
3.2 CDR 服务器......11
3.2.1 CDR 服务器功能描述......11
3.2.2 CDR 服务器设计框架......12

4. 远程单元......18

5. 技术特点......20
5.1 REST 风格的 Web API 接口......20
5.2 nano-SOA 架构......21
5.2.1 SOA vs. AIO......21
5.2.2 nano-SOA 架构......22
5.2.3 白杨消息端口交换服务（BYPSS）......23
5.2.4 白杨分布式消息队列（BYDMQ）......38
5.3 安全隧道服务（BYST）......43

6. 总结 46

1. 概述

1.1 行业背景

在通信技术日益普及，通信成本日益下降的今日，各企事业单位对功能完善、运行稳定的话单计费与统计系统的需求正日益迫切。

随着 IP 网络的不断普及，企业正越来越多地使用 VoIP 中继线路在自己的各个分支机构间实现互联（IPT）。加上异种 PBX 环境的日益增加，使得每个呼叫都可能经过复杂的路由路径。这样的环境与以往的单点计费应用有着本质区别：首先，计费软件必须能够有效区分网内与网外等不同通讯方式，并分别予以计费。其次，使用了 IPT 系统的企业，除了需要计算实际话费外，几乎都要求加入参考话费的计算能力，以便进一步核算新的 IPT 系统带来的话费节省。对以上两种多点 PBX 组网的典型需求，传统的，基于单点的计费软件均无法满足。

此外，随着传统电信运营商和虚拟（IP）运营商正逐渐增多。为了吸引客户，各运营商也都推出了自己的费率价格和优惠政策。现在的企业越来越倾向于组合使用多个运营商的服务，以达到最低成本。很显然，传统计费软件的单运营商模式无法满足这类需求。

还应当看到，在当今贸易全球化的大趋势下，越来越多的公司的企业需要在世界各个地区开设自己的分支机构。由于大多数运营商都在不同时段和节假日使用不同的费率价格，这就使得传统的单时区、单夏令时规则、单节假日定义的计费模式不再适用。

与此同时，随着 LINUX、BSD 系列、以及 OpenSolaris、Darwin 等各类优秀开源操作系统的不断涌现、完善与普及，加之 Oracle、IBM、HP、Cisco、Avaya 等业界领先厂商对这些新兴操作系统所给与的积极支持与重视。如今各企业内的服务器环境早已不再是 x86+WinNT 一统天下的时代，在这些环境中或多或少地存在着各类异种平台的身影。

在以上各前提之下，计费系统作为企业的关键服务之一，在保证运行稳定、工作可靠、功能完善、处理高效的基础上，也必须提供良好的可移植性和跨平台互联能力。由此可见，一种支持多站点、多 SP、多时区以及多 PBX 的跨平台计费系统已势在必行。

上海重玄科技凭借多年行业经验和雄厚的研发实力，从客户的实际需求出发，经多年不断完善，成功推出了新一代计费与成本管理系统。

1.2 系统描述

蓝鲸计费在企业通信系统中的位置如下图所示：

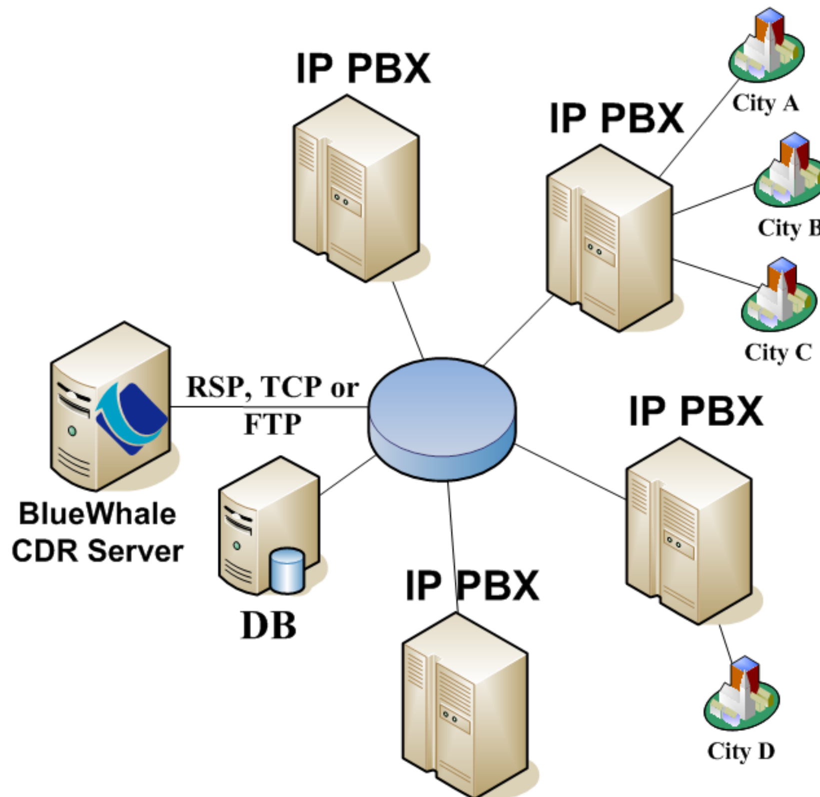


图 1

蓝鲸计费系统能够支持目前大多数 PBX 的计费方式，特别针对大型 PBX 进行设计。对于繁忙的中继线路，以及大量的内部分机，蓝鲸系统都可以从容计费。此外，针对当前越来越多企业选择自己搭建基于 IPT 的呼叫中心之现状，蓝鲸计费系统能够同时支持呼入计费（400/800 业务）、呼出计费和内部计费，并且可以为每种计费方式设置独立的话费规则。

针对 Avaya Communication Manager，蓝鲸计费系统不但支持以 TCP 方式进行话单接收和精确到秒的高精准计费，还能够以 RSP（可靠会话层协议）连接方式收取话单，增强了计费精度和系统可靠性。

对于 Cisco CallManager，蓝鲸计费系统能够智能分析电话会议（Conference Call）、呼叫转移（Call Transfer）、自动转接（Call Forwarding）、cBarge 等各类复杂的话单序列，以及他们之间的各种组合呼叫。

为方便管理员的日常监控，蓝鲸计费系统具备实时计费（实时话单分析与批价）和话单流水功能。同时，由于各企业的统计需求千差万别，计费系统还带有基于插件形式，可灵活配置的数据统计与分析模块。以便根据企业的不同需求实现定制插件，帮助企业更好地掌握成本分布。

作为一款企业级应用，蓝鲸计费系统能够可靠地运行于 Microsoft Windows NT/2k/2k3、POSIX（UNIX / LINUX / Solaris）以及 Apple Macintosh 等多种常见服务器平台中。在稳定性方面，计费系统能够承受上百 PBX，每秒数百万峰值话单的超压测试。同时解决了以往的 IP 计费系统中，经常发生的内存泄漏、CPU 占用过高以及话单异常等各类问题。

蓝鲸计费系统真正作到了对分布式PBX的支持，可以有效地区分Multi-Location，能够良好支持多站点应用。可以为不同地点、不同运营商分别配置时区、夏令时及优惠时段等信息。可以捕捉到不同站点之间的分机间的通话，并且对其进行记录。在PBX支持相应字段的前提下，可以有效识别和记录每条话单的来源IP地址、PBX ID以及中继线路等标识信息，从而真正作到对多PBX的支持。

蓝鲸计费系统能够应客户要求支持MySQL、MS SQL Server、PostgreSQL、ORACLE、DB2等各种主流数据库，同时内置了一套可处理高达数千万条话单、TB量级的SQLite嵌入式数据库引擎，用以作为脱机数据库（Offline DB）和主数据库的被选方案。通过使用多级话单内存解析等技术，系统能够在瞬间负载数百万条话单（通常发生在多台大型PBX同时突发性吐出大量缓冲话单数据时）。对于今天的企业级应用来说，产品的稳定性和它所能支持的数据库接口都属至关重要的考量因素。

一个优秀的计费系统需要从各个方面保护企业数据，蓝鲸计费通过使用脱机数据库（Offline DB）技术保证：即使企业的主数据库服务器由于维护和故障等原因临时拓机，也不会产生任何数据丢失。计费系统将使用内嵌的脱机数据库存储和管理主数据库拓机期间接收并分析入库的所有话单数据。当主数据库恢复到可用状态时，这些信息将被自动提交。蓝鲸计费系统的嵌入式数据库引擎可以有效缓存和管理高达数千万条CDR数据。

另外，对于Avaya等易失性话单源，计费系统还可以通过RAW DUMP机制对原始话单进行持久化保存，以便管理员核查和在需要时重放。

除此之外，为了满足电信级计费需求，计费系统还提供了在线备份与恢复、异地容灾、双机容错等7×24h高可用性支持，通过合理使用异地容灾和双机容错功能，即使在地震、火灾等灾难性事故面前，仍有可能做到零拓机时间，确保数据安全。

企业计费的最终目标就是要生成各种格式和不同详细程度的统计报表。丰富强大的统计功能对于任何一个成功的计费系统来说都是不可或缺的要素。但企业间的统计需求千差万别，蓝鲸计费系统不但支持常见的信息查询和过滤选项，更支持为不同用户定制专用统计分析插件的功能。

与此同时，查询和统计分析的结果可以通过Excel、HTML、CSV、TXT、JSON等常用格式导出，便于与其它软件交互及进一步分析、排版和打印。

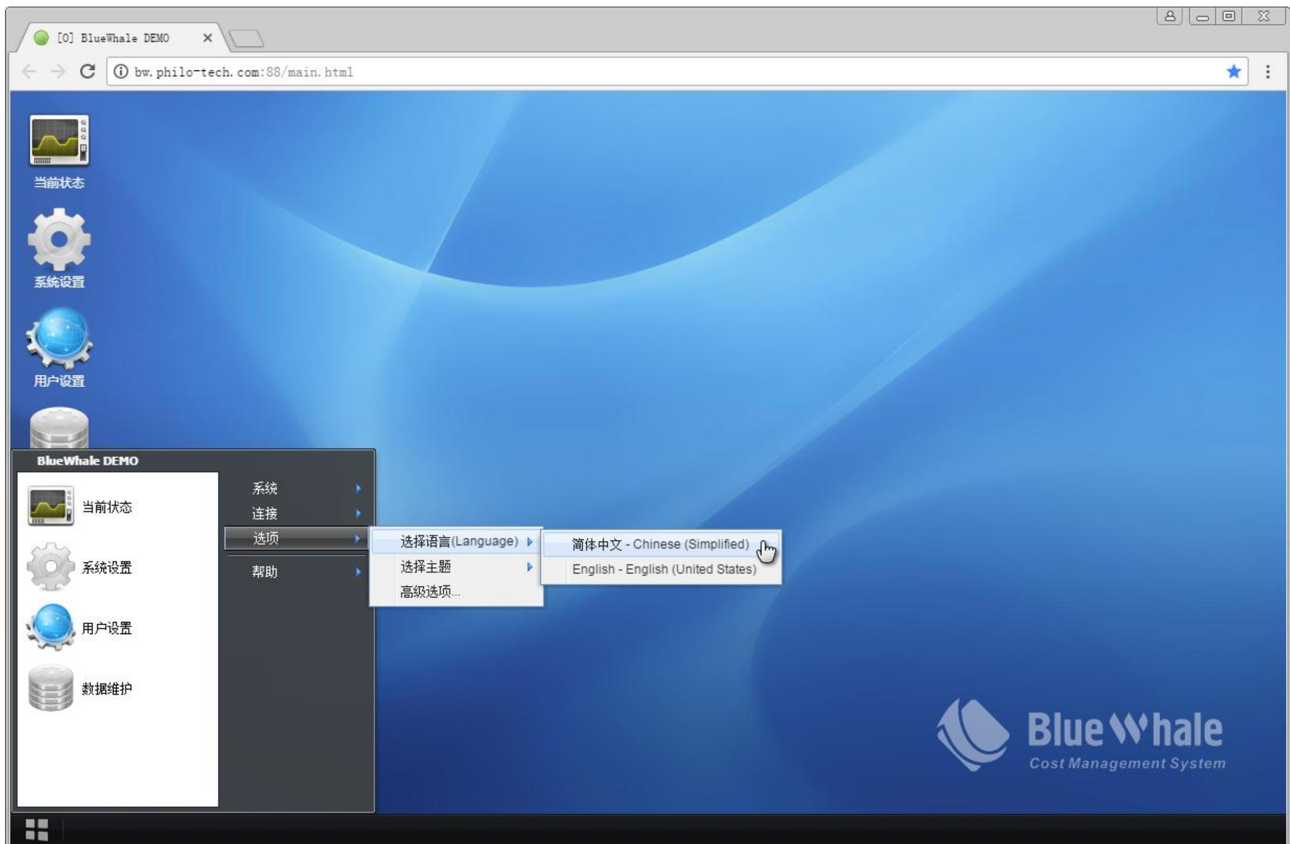


图 2 蓝鲸计费系统管理界面

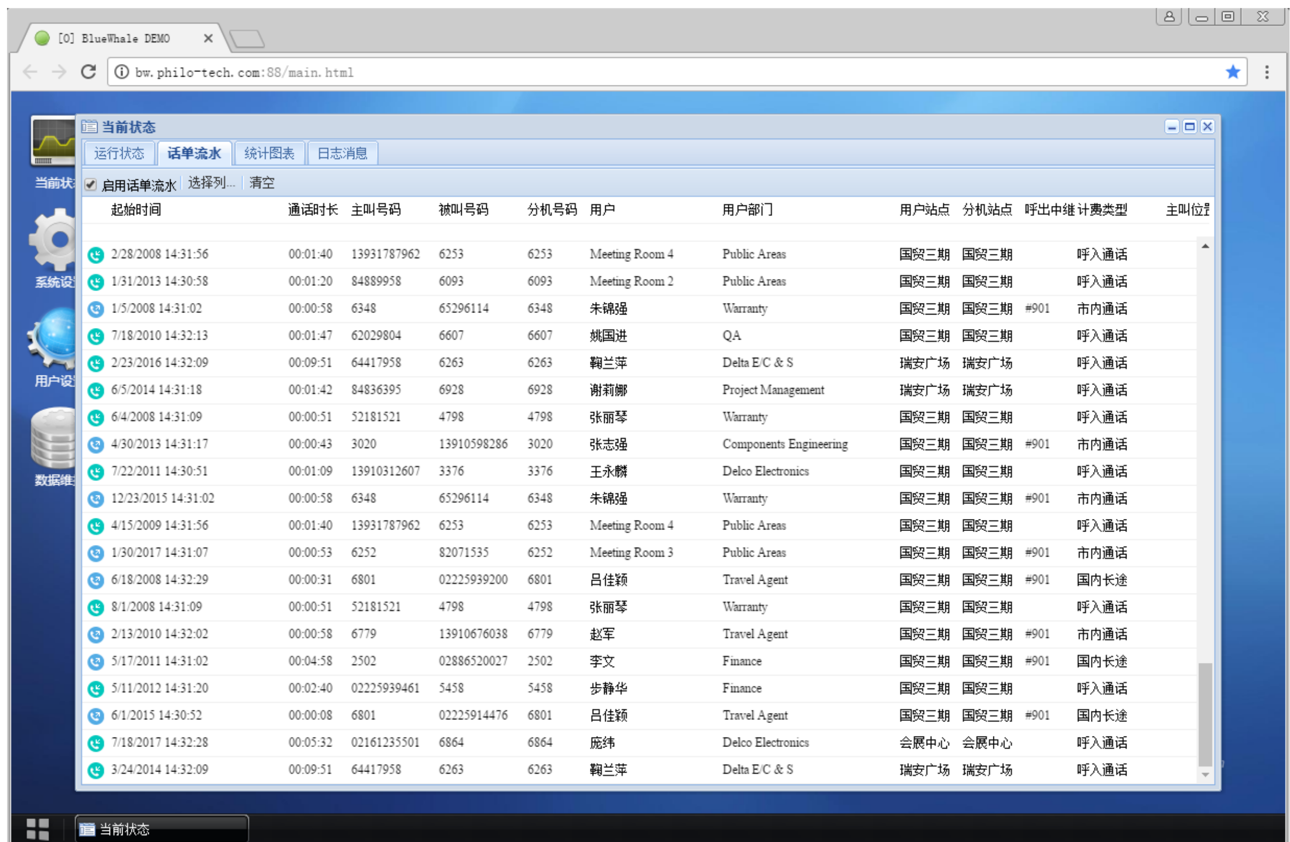


图 3 蓝鲸计费系统管理界面

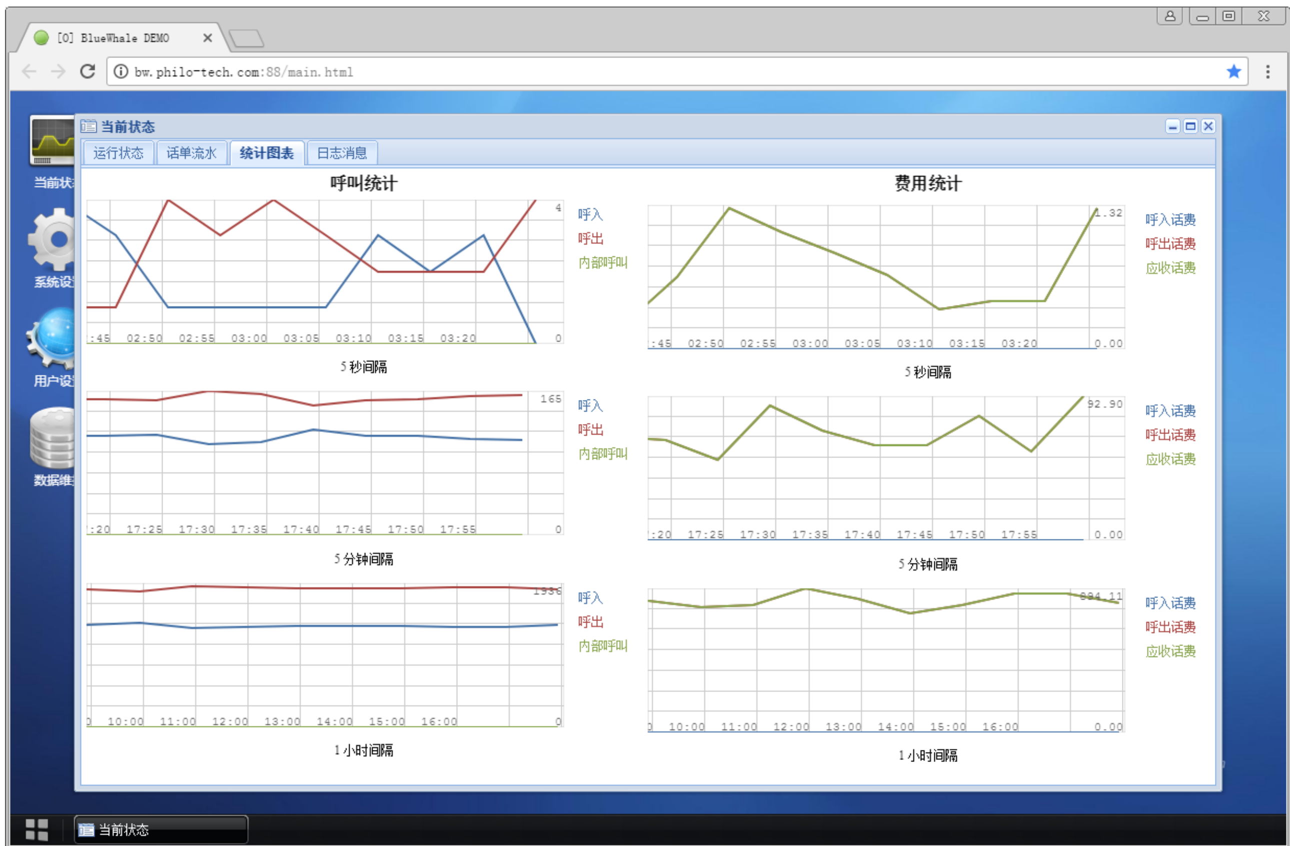


图 4 蓝鲸计费系统管理界面

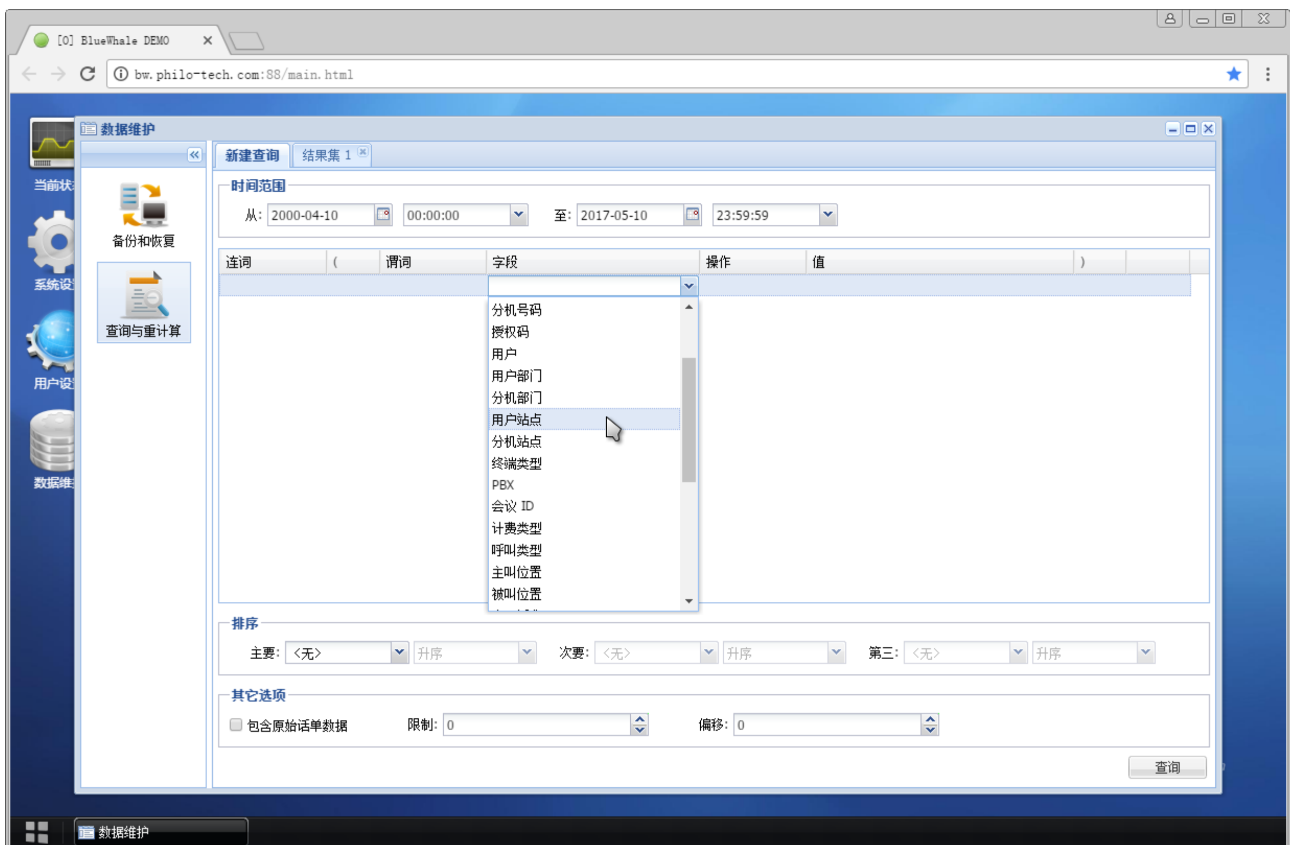


图 5 蓝鲸计费系统管理界面

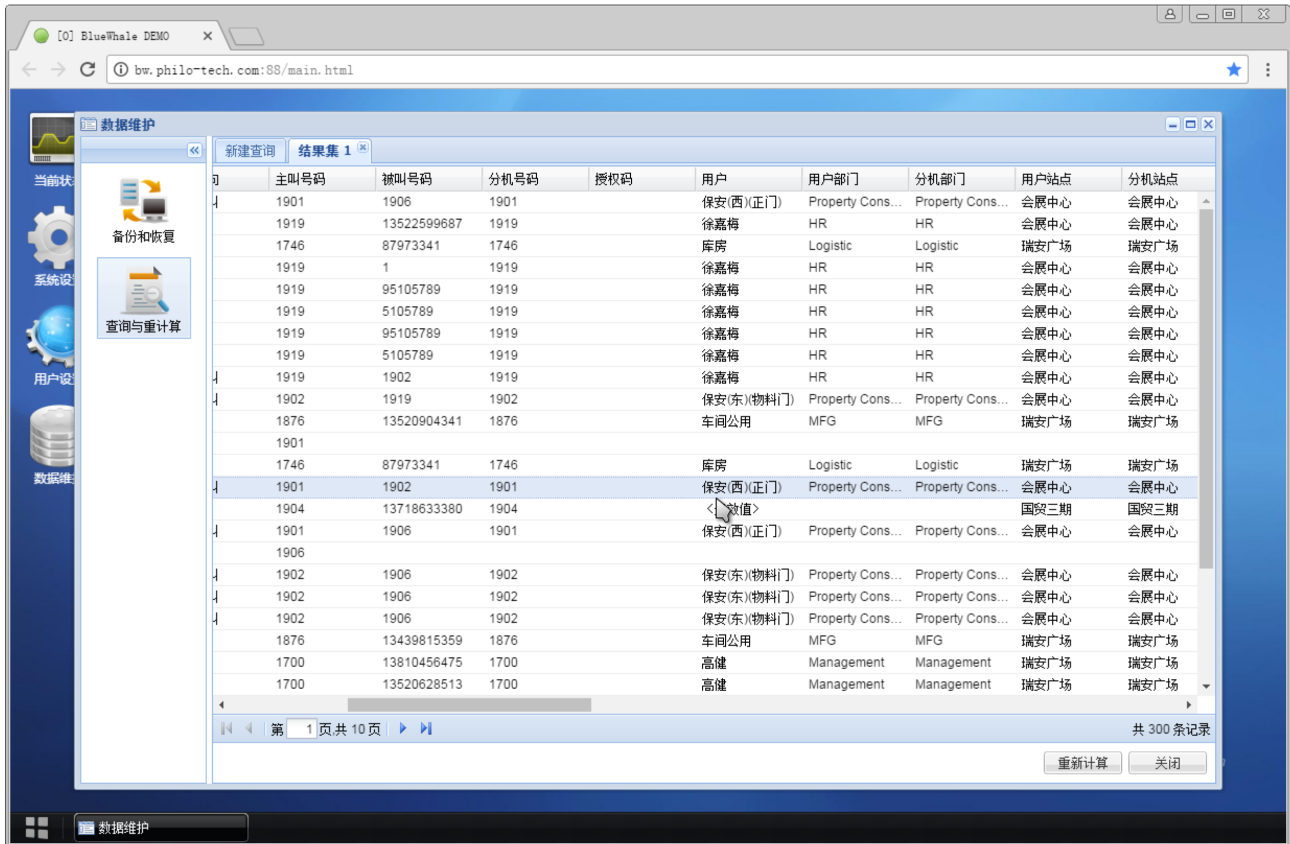


图 6 蓝鲸计费系统管理界面

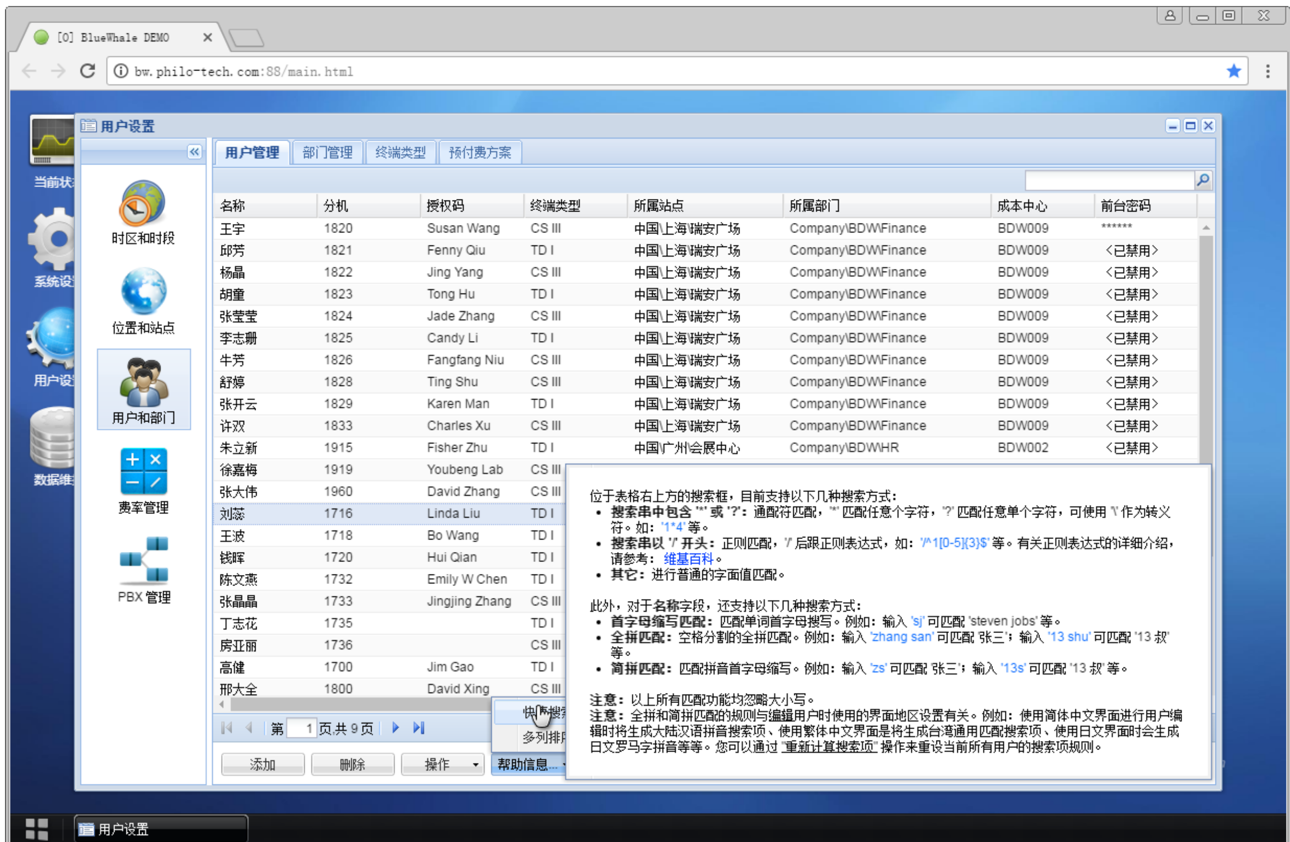


图 7 蓝鲸计费系统管理界面

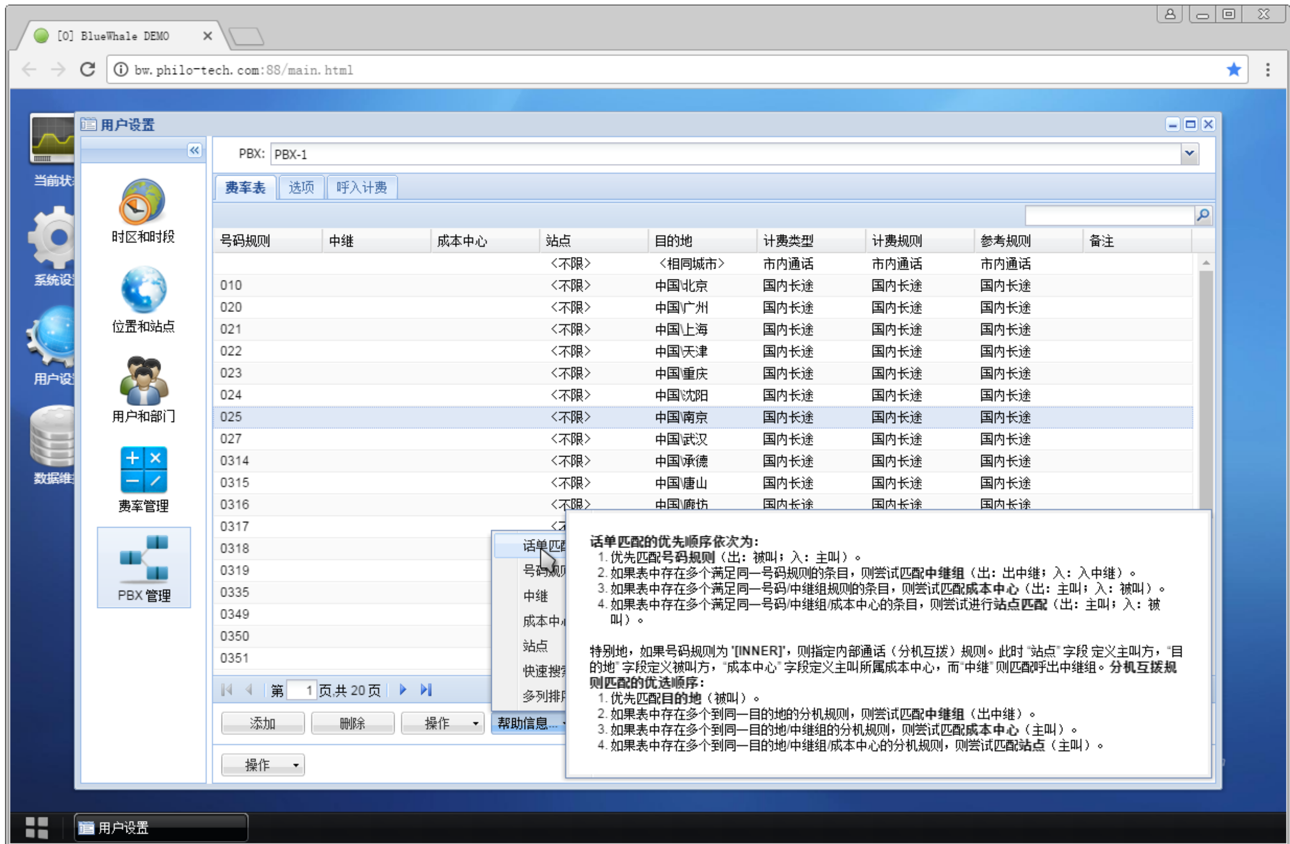


图 8 蓝鲸计费系统管理界面

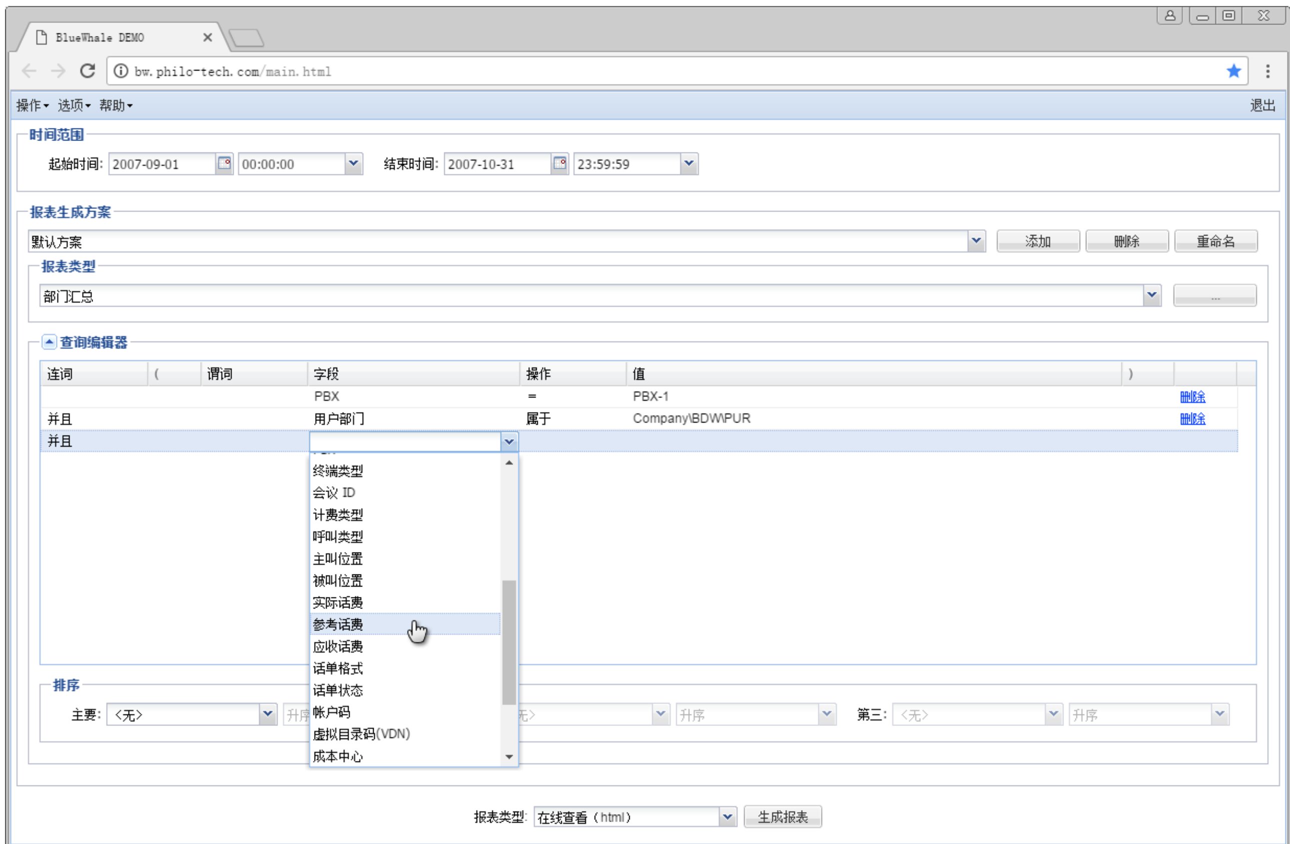
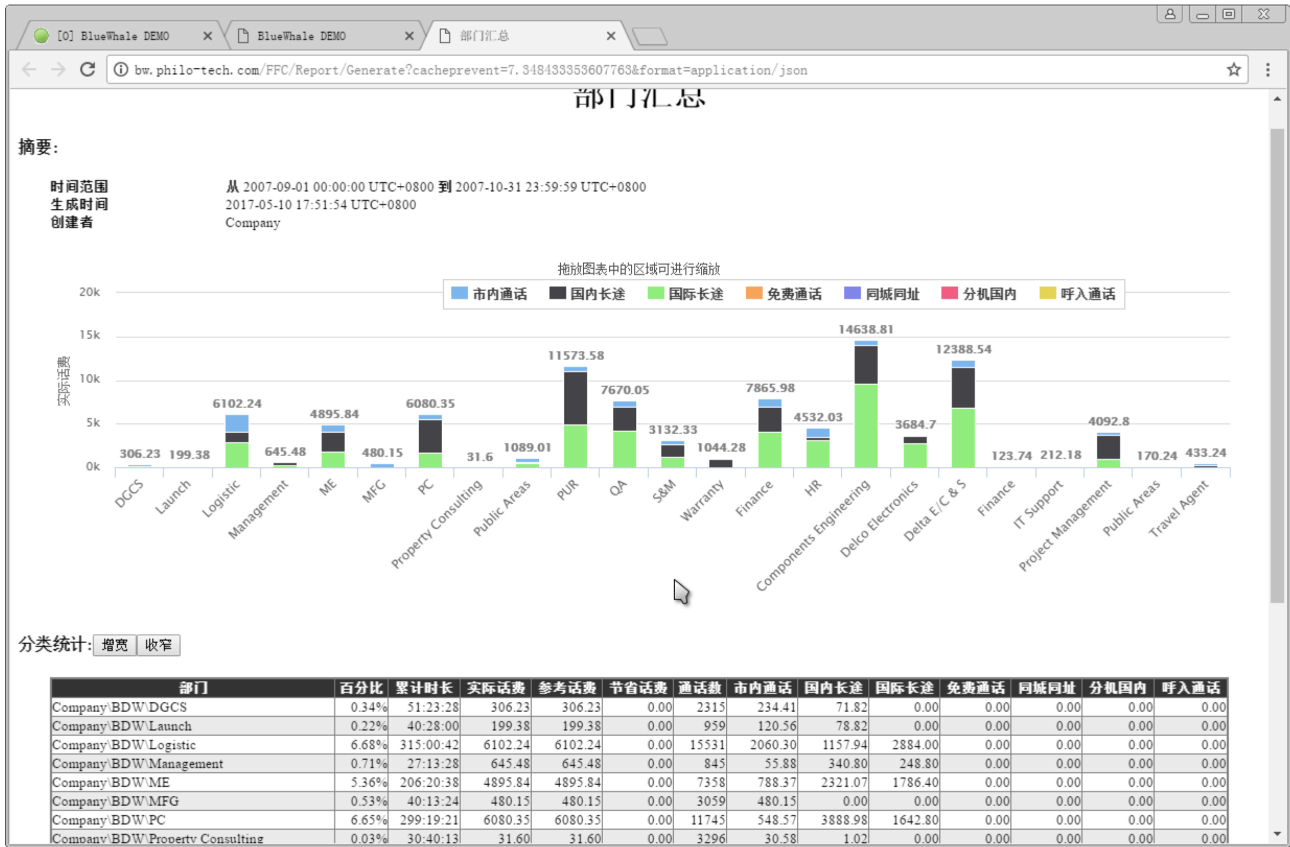


图 9 蓝鲸计费系统报表生成界面



部门	百分比	累计时长	实际话费	参考话费	节省话费	通话数	市内通话	国内长途	国际长途	免费通话	同城同址	分机国内	呼入通话
Company\BDW\DGCS	0.34%	51:23:28	306.23	306.23	0.00	2315	234.41	71.82	0.00	0.00	0.00	0.00	0.00
Company\BDW\Launch	0.22%	40:28:00	199.38	199.38	0.00	959	120.56	78.82	0.00	0.00	0.00	0.00	0.00
Company\BDW\Logistic	6.68%	315:00:42	6102.24	6102.24	0.00	15531	2060.30	1157.94	2884.00	0.00	0.00	0.00	0.00
Company\BDW\Management	0.71%	27:13:28	645.48	645.48	0.00	845	55.88	340.80	248.80	0.00	0.00	0.00	0.00
Company\BDW\ME	5.36%	206:20:38	4895.84	4895.84	0.00	7358	788.37	2321.07	1786.40	0.00	0.00	0.00	0.00
Company\BDW\MFG	0.53%	40:13:24	480.15	480.15	0.00	3059	480.15	0.00	0.00	0.00	0.00	0.00	0.00
Company\BDW\PC	6.65%	299:19:21	6080.35	6080.35	0.00	11745	548.57	3888.98	1642.80	0.00	0.00	0.00	0.00
Company\BDW\Property Consulting	0.03%	30:40:13	31.60	31.60	0.00	3296	30.58	1.02	0.00	0.00	0.00	0.00	0.00

图 10 蓝鲸计费系统报表示例

图 11 蓝鲸计费系统报表示例

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	部门综合统计													
2	时间范围	从 2000-10-09 00:00:00 UTC+0800 到 2012-11-09 23:59:59 UTC+0800												
3	生成时间	2012-11-09 15:49:31 UTC+0800												
4	创建者	Company												
5														
6	呼叫总计	121911												
7	话费总计	87972.78												
8	总计节省	0.00												
9														
10	部门	用户	实际话费	节省话费	通话数	累计时长	部门累计	市内通话	国内长途	国际长途	免费通话	同城同址	分机国内	呼入通话
11	Company\BDW\DGCS	[illegible]	74.25	0.00	291	0D 08:18:45	-	15.62	58.63	0.00	0.00	0.00	0.00	0.00
12	Company\BDW\DGCS	[illegible]	180.38	0.00	1176	0D 21:13:28	-	146.08	34.30	0.00	0.00	0.00	0.00	0.00
13	Company\BDW\DGCS	[illegible]	56.87	0.00	447	0D 12:06:22	-	56.87	0.00	0.00	0.00	0.00	0.00	0.00
14	Company\BDW\DGCS	[illegible]	22.55	0.00	432	0D 10:14:57	-	22.55	0.00	0.00	0.00	0.00	0.00	0.00
15	Company\BDW\DGCS	-	-	-	-	-	334.05	-	-	-	-	-	-	-
16	Company\BDW\Launch	[illegible]	209.49	0.00	969	1D 16:27:55	-	123.53	85.96	0.00	0.00	0.00	0.00	0.00
17	Company\BDW\Launch	-	-	-	-	-	209.49	-	-	-	-	-	-	-
18	Company\BDW\Logistic	[illegible]	907.51	0.00	560	0D 13:25:45	-	50.16	55.75	801.60	0.00	0.00	0.00	0.00
19	Company\BDW\Logistic	[illegible]	605.05	0.00	1351	1D 17:14:25	-	122.43	482.62	0.00	0.00	0.00	0.00	0.00
20	Company\BDW\Logistic	[illegible]	80.86	0.00	692	0D 11:45:37	-	29.70	51.16	0.00	0.00	0.00	0.00	0.00
21	Company\BDW\Logistic	[illegible]	350.42	0.00	891	1D 00:32:53	-	81.18	269.24	0.00	0.00	0.00	0.00	0.00
22	Company\BDW\Logistic	[illegible]	243.73	0.00	558	0D 14:55:42	-	34.76	208.97	0.00	0.00	0.00	0.00	0.00
23	Company\BDW\Logistic	[illegible]	74.39	0.00	736	0D 11:35:38	-	67.65	6.74	0.00	0.00	0.00	0.00	0.00
24	Company\BDW\Logistic	[illegible]	43.11	0.00	427	0D 06:27:51	-	39.93	3.18	0.00	0.00	0.00	0.00	0.00
25	Company\BDW\Logistic	[illegible]	100.73	0.00	673	0D 16:01:32	-	96.91	3.82	0.00	0.00	0.00	0.00	0.00
26	Company\BDW\Logistic	[illegible]	105.20	0.00	524	0D 16:45:12	-	100.98	4.22	0.00	0.00	0.00	0.00	0.00
27	Company\BDW\Logistic	[illegible]	61.38	0.00	406	0D 08:33:24	-	49.83	11.55	0.00	0.00	0.00	0.00	0.00
28	Company\BDW\Logistic	[illegible]	1198.48	0.00	6613	4D 13:49:31	-	1160.06	38.42	0.00	0.00	0.00	0.00	0.00
29	Company\BDW\Logistic	[illegible]	2450.27	0.00	1912	1D 12:49:52	-	253.55	114.32	2082.40	0.00	0.00	0.00	0.00
30	Company\BDW\Logistic	[illegible]	68.81	0.00	365	0D 05:33:07	-	28.82	39.99	0.00	0.00	0.00	0.00	0.00
31	Company\BDW\Logistic	-	-	-	-	-	6289.94	-	-	-	-	-	-	-
32	Company\BDW\Management	[illegible]	144.19	0.00	395	0D 11:53:27	-	33.55	110.64	0.00	0.00	0.00	0.00	0.00
33	Company\BDW\Management	[illegible]	492.06	0.00	377	0D 12:56:12	-	18.15	225.11	248.80	0.00	0.00	0.00	0.00
34	Company\BDW\Management	[illegible]	9.75	0.00	76	0D 01:31:26	-	6.60	3.15	0.00	0.00	0.00	0.00	0.00
35	Company\BDW\Management	-	-	-	-	-	646.00	-	-	-	-	-	-	-

图 12 蓝鲸计费系统报表示例

1.3 技术指标

蓝鲸计费系统使用了先进的多级内存解析配合多线程并发技术，既挖掘了多核系统的巨大性能潜力又提高了系统整体可靠性和峰值容限能力。除此之外，系统关键模块中还大量应用了哈希表、红黑树、LRU Cache 和多级平衡键树等高效数据结构，以及多线程安全的内存零拷贝传递（基于原子量的引用计数和写时拷贝）、延迟计算和表达式预编译等各种高效算法进一步提高了计费系统的各项性能指标。

若无特殊说明，以下所有实测数据均取自配置为：Intel Xeon E5606（2GHz，4 核 8 线程）、8GB Mem、Win2k3 32bit Version 的测试环境；标有"64bit"字样的指标则取自配置为：Intel Xeon E5606（2GHz，4 核 8 线程）、16GB Mem、Win2k8 64bit Version 的测试环境。

* **性能指标**：
  - 32bit Version：17 万条话单每秒的持续处理能力，160 万条话单每秒的峰值处理能力（持续 1 分钟），可通过增加内存来提高峰值处理能力。
  - 64bit Version：19 万条话单每秒的持续处理能力，350 万条话单每秒的峰值处理能力（持续 1 分钟），通过增加内存可有效提高峰值处理能力。
* **用户（分机）容限**：设计容限：20 万；最高容限：100 万。
* **站点（出端口）容限**：设计容限：5 万；最高容限：15 万。
* **PBX 并发连接**：设计容限：200 PBX 并发连接；最高容限：1000 并发。
* **运营商数量**：设计容限：300；最高容限：1000。以上数据为每家运营商配置 3000 条计费规则时的测试结果。
* **计费规则数量**：设计容限：10 万条；最高容限：100 万条。
* **脱机数据库（Offline DB）容限**：设计容限 1000 万条话单，1TB；最高容限：2000 万条话单，2TB。
* **主数据库（Main DB）容限**：由具体数据库产品决定。内置的 SQLite 嵌入式数据库引擎可支持：设计容限 1000 万条话单，1TB；最高容限：2000 万条话单，2TB。
* **平台兼容性**：目前已通过测试的兼容平台包括：
  - 32BIT Microsoft Windows NT Kernel（Windows 2000、Windows XP、Windows Server 2003、Windows Vista and Windows Server 2008）。
  - 64BIT Microsoft Windows（Windows XP 64bit Edition、Windows Server 2003 64bit

Edition、Windows Vista 64bit Edition and Windows Server 2008 64bit Edition、Windows Server 2008r2）。

- Linux Kernel 2.6.x/3.x：32BIT and 64BIT Version。
- FreeBSD 6.x/7.x/8.x 32BIT and 64BIT Version。
- OpenSolaris Version 0906。

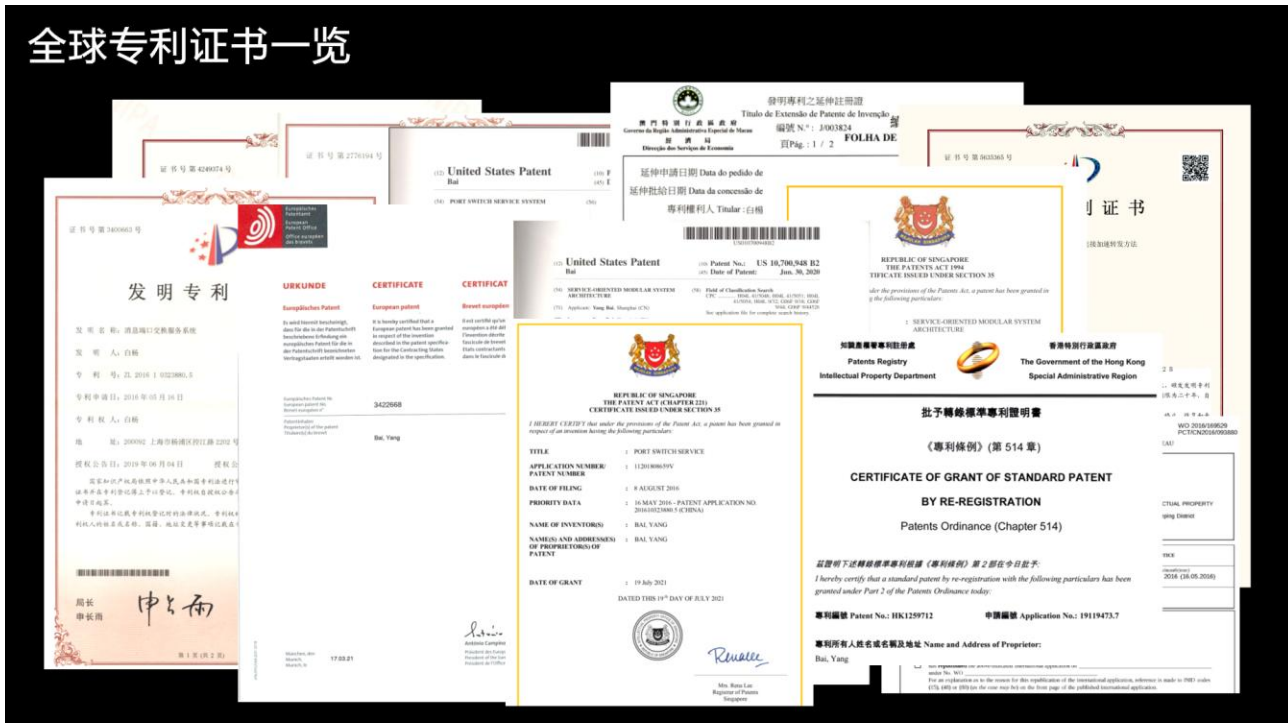
注：蓝鲸计费系统目前支持的硬件平台包括：x86/x64, IA64, ARM, RISC-V, MIPS, POWER 以及 SPARC 架构。

1.4 应用支撑平台

《白杨应用支撑平台》作为我方的核心竞争力，使用汇编、C/C++ 构建，包含数百万行代码和上千个成熟的通用组件。上述单点高并发和多活 IDC 分布式集群等核心技术均由白杨应用支撑平台提供。多年来，基于支撑平台的各种产品已被广泛部署于包括兴业银行（China CIB）、中石油（CNPC）、华安保险（Sinosafe Insurance）、淘宝网（taobao.com）、易果网（yiguo.com）、烟台万华集团、法国兴业银行（SOCIETE GENERALE）、中国农业银行（ABC）、宝马集团（BMW）、陕汽集团（陕汽重卡）、德尔福汽车（Delphi）、美联航（United Airlines）、GE（美国通用电气）、贝塔斯曼（Bertelsmann）、埃森哲（Accenture）、光大银行（CEB）、壹基金（One Foundation）、中国宋庆龄基金会、中国移动（China Mobile）、中国联通（China Unicom）、国家电网（SGCC）等各大企业在内的不同生产环境中：

真实生产环境下的大范围部署不但为上层应用提供了可靠的、平台无关的底层环境，也进一步检验了支撑平台的可靠性、稳定性、可移植性、高效性等各方面指标。

支撑平台使用汇编、C/C++ 构建，支持 Windows、Linux、BSD、IBM AIX、HP-UX、Solaris、MAC OS X、vxWorks、QNX、DOS, WinCE (Windows Mobile), NanoGUI、eCos、RTEMS 以及 Android、iOS 等绝大多数主流操作系统。支持 x86/x64、ARM、IA64、MIPS、RISC-V、POWER、SPARC 等主流硬件平台。在提供了大量高品质可重用组件的同时，也确保了良好的可移植性。



上千成熟可靠的高品质功能组件可在性能、功能和稳定性等方面大幅提升软件产品的品质，并为产品的开发带来了难以想象的便利，例如：

1.4.1 单点支撑千万量级并发的高效 IO 服务器组件

支撑平台使用汇编和异步 IO 对网络服务组件进行了优化，通过 DMA + 硬件中断实现内存零拷贝的高效异步网络服务。性能、可靠性和可伸缩性都很强。可在 2011 年出厂的，当时售价不足 2 万元人民币的单台至强 5600 系入门级 1U PC Server 上支撑上千万 TCP/HTTP 并发连接。相对应地，一般由 Java / .NET 开发的服务器端，在相同配置的机器上，单点最高仅可支撑 3000 到 5000 并发，PHP 则更低（具体可参考《白杨应用支撑平台技术白皮书》：3.2.1、3.3.1 以及 3.3.2 小节）。

1.4.2 强一致多活 IDC 高可用（HAC）和高性能（HPC）服务器集群

强一致、抗脑裂（Split Brain）的多活 IDC 分布式高可用（HAC）和高性能（HPC）计算集群支持：独有的 nano-SOA 大规模分布式架构在保持高内聚、低耦合设计的前提下，将单点性能

提升到了远超传统 SOA 架构的水平，同时简化了集群部署，提高了集群的可维护性。

白杨消息端口交换服务（BYPSS）：一种基于多数派算法的，强一致（抗脑裂）、高可用的分布式协调组件，可用于向集群提供服务发现、故障检测、服务选举、分布式锁等传统分布式协调服务，同时还支持消息分发与路由等消息中间件功能。由于通过专利算法消除了传统 Paxos/Raft 中的网络广播和磁盘 IO 等主要开销，再加上批量模式支持、并发散列表、高并发服务组件等大量其它优化，使得 BYPSS 可在延迟和吞吐均受限的跨 IDC 网络环境中支持百万节点、万亿端口量级的超大规模计算集群。

带强一致保证的多活 IDC 技术是现代高性能和高可用集群的关键技术，也是业界公认的主要难点。作为实例：2018 年 9 月 4 日微软美国中南区某数据中心空调故障导致 Office、Active Directory、Visual Studio 等服务下线近 10 小时不可用；2015 年 8 月 20 日 Google GCE 服务中断 12 小时并永久丢失部分数据；2015 年 5 月 27 日、2016 年 7 月 22 及 2019 年 12 月 5 日支付宝多次中断数小时；以及 2013 年 7 月 22 日、2023 年 3 月 29 日微信服务中断数小时等重大事故均属于产品未能实现多活 IDC 架构，单个 IDC 故障导致服务全面下线的惨痛案例。

在上述方面，我方拥有超过十年的积累，掌握多项受国家和国际发明专利保护的分布式架构和算法。得益于这些领先的强一致、高可用、高性能分布式集群算法和架构，我们在蓝鲸、白豚、职业精等全线产品上，均实现了真正的多活 IDC 架构，为客户提供了无以伦比的数据可靠性和服务可用性保证。

1.4.2.1 分布式协调服务

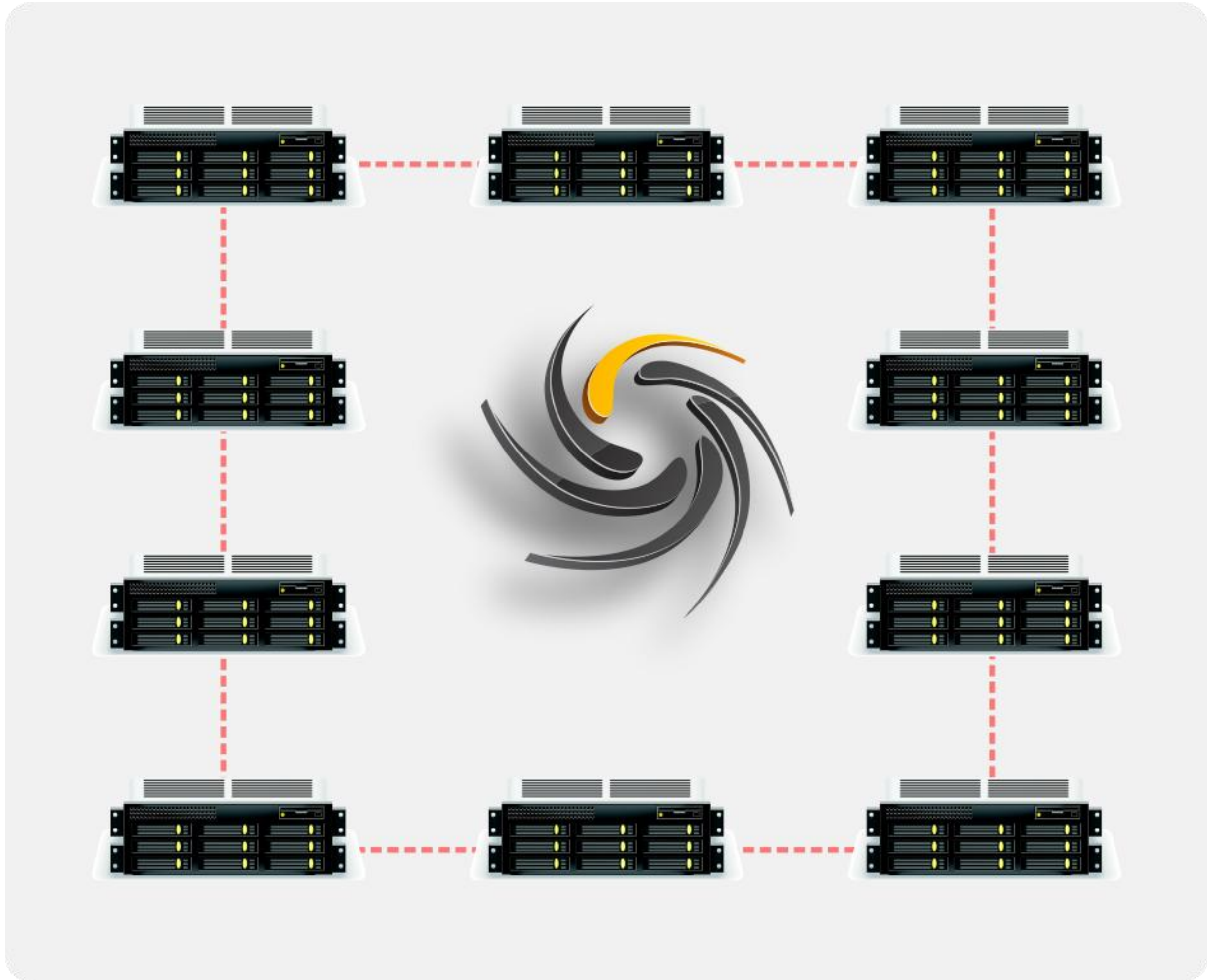


图 13

分布式协调服务为集群提供服务发现、服务选举、故障检测、故障转移、故障恢复、分布式锁、任务调度，以及消息路由和消息分发等功能。

分布式协调服务是分布式集群的大脑，负责指挥集群中的所有服务器节点协同工作。将分布式集群协调为一个有机整体，使其有效且一致地运转。实现可线性横向扩展的高性能（HPC）和高可用（HAC）分布式集群系统。

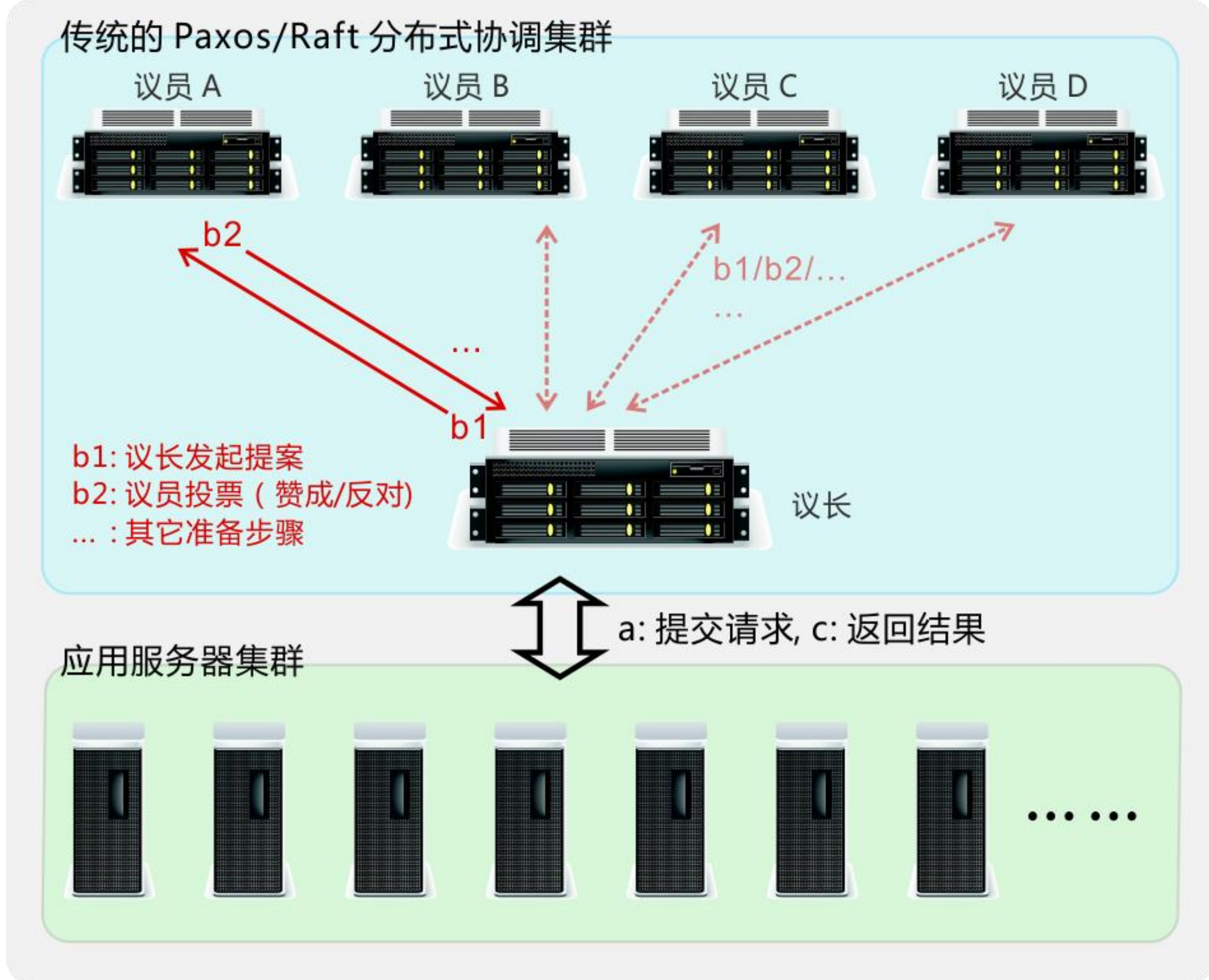


图 14

传统的 Paxos/Raft 分布式协调算法为每个请求发起投票，产生至少 2 到 4 次网络广播（b1、b2、···）和多次磁盘 IO。使其对网络吞吐和通信时延要求很高，无法部署在跨 IDC（城域网或广域网）环境。

我们的专利算法则完全消除了此类开销。因此大大降低了网络负载，显著提升整体效率。并使得集群跨 IDC 部署（多活 IDC）变得简单可行。

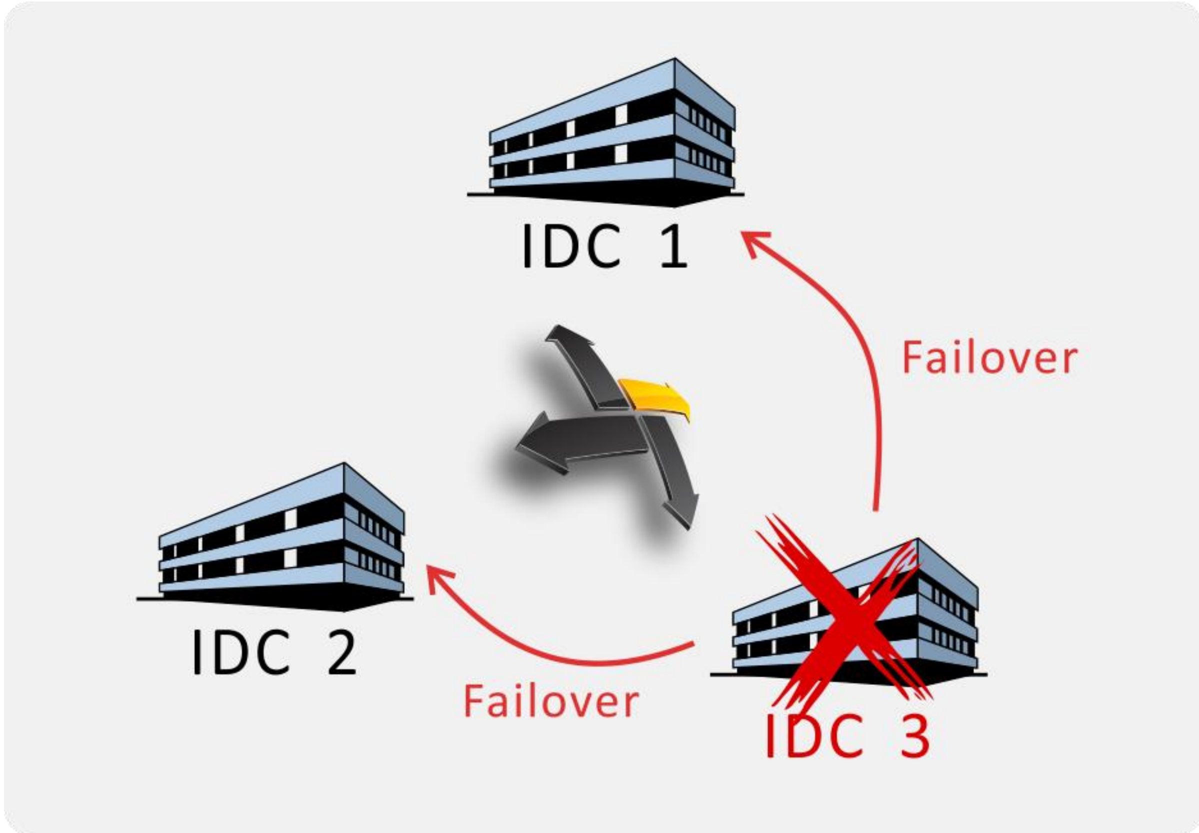


图 15

基于我方独有的分布式协调技术，可实现高性能、强一致的多活 IDC 机制。可在毫秒级完成故障检测和故障转移，即使整座 IDC 机房下线，也不会导致系统不可用。同时提供强一致性保证：即使发生了网络分区也不会出现脑裂（Split Brain）等数据不一致的情形。例如：

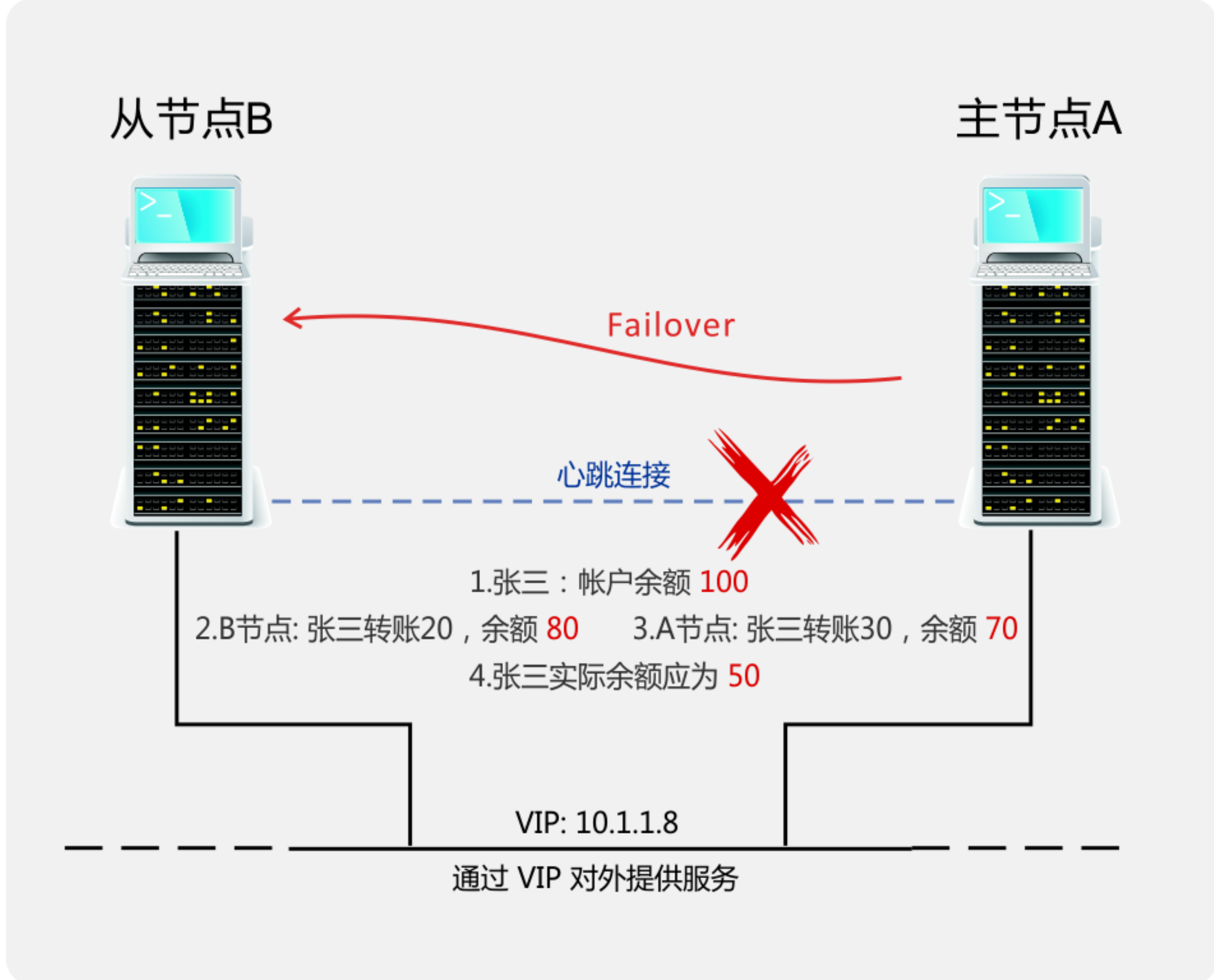


图 16

在传统的双机容错方案中，从节点在丢失主节点心跳信号后，会自动将自身提升为主节点，并继续对外提供服务，以实现高可用。在此种情形中，当主从节点均正常，但心跳连接意外断开时（网络分区），就会发生脑裂（Split Brain）问题，如图 16 所示：此时 A、B 均认为对方已下线，故将自己提升为主节点并分别对外提供服务，产生难以恢复的数据不一致。

我方 BYPSS 服务可提供与传统 Paxos/Raft 分布式算法相同水平的强一致性保证，从根本上杜绝脑裂问题的发生。

类似地：工行、支付宝等服务也有异地容灾方案（支付宝：杭州 → 深圳、工行：上海 → 北京）。但在其异地容灾方案中，两座 IDC 之间并无 Paxos 等分布式协调算法保护，因此无法实现强一致，也无法避免脑裂。

举例来说，一个在支付宝成功完成的转账交易，可能要数分钟甚至数小时后才会从杭州主 IDC 被异步地同步到深圳的灾备中心。杭州主 IDC 发生故障后，若切换到灾备中心，意味着这些未同步的交易全部丢失，并伴随大量的不一致。比如：商家明明收到支付宝已收款提示，并且在淘宝交易系统看到买家已付款，并因此发货。但由于灾备中心切换带来的支付宝交易记录丢失，导致在支付宝中丢失了相应的收入，但淘宝仍然提示买家已付款。因此，工行、支付宝等机构在主 IDC 发生重大事故时，宁可停止服务几个小时甚至更久，也不愿意将服务切换到灾备中心。只有在主

IDC 发生大火等毁灭级事故后，运营商才会考虑将业务切换到灾备中心（这也是灾备中心建立的意义所在）。

因此，异地容灾与我方的强一致、高可用、抗脑裂多活 IDC 方案具有本质区别。

此外，Paxos / Raft 在经历过半节点同时故障下线并维修恢复的过程中，无法保证数据的强一致性，可能产生幻读等不一致问题（例如：在一个三节点集群中，节点 A 因为电力故障下线，一小时后节点 B 和 C 则因为磁盘故障下线。此时节点 A 恢复电力供应重新上线，紧接着管理员更换了节点 B 和 C 的磁盘并让它们分别恢复上线。此时整个集群 1 小时内的修改将全部丢失，集群退回到了 1 小时前 A 节点下线时的状态）。而 BYPSS 则从根本上避免了此类问题的发生，因此 BYPSS 拥有比 Paxos / Raft 更强的一致性保证。

由于消除了 Paxos/Raft 算法中的大量广播和分布式磁盘 IO 等高开销环节，配合支撑平台中的高并发网络服务器、以及并发散列表等组件。使得 BYPSS 分布式协调组件除了上述优势外，还提供了更多优秀特性：

批量操作：允许在每个网络包中，同时包含大量分布式协调请求。网络利用率极大提高，从之前的不足 5% 提升到超过 99%。类似于一趟高铁每次只运送一位乘客，与每班次均坐满乘客之间的区别。实际测试中，在单千兆网卡上，可实现 400 万次请求每秒的性能。在当前 IDC 主流的双口万兆网卡配置上，可实现 8000 万次请求每秒的吞吐。比起受到大量磁盘 IO 和网络广播限制，性能通常不到 200 次请求每秒的 Paxos/Raft 集群，有巨大提升。

超大容量：通常每 10GB 内存可支持至少 1 亿端口。在一台插满 64 根 DIMM 槽的 1U 尺寸入门级 PC Server 上（8TB），可同时支撑至少 800 亿对象的协调工作；在一台 32U 大型 PC Server 上（96TB），可同时支撑约 1 万亿对象的分布式协调工作。相对地，传统 Paxos/Raft 算法由于其各方面限制，通常只能有效管理和调度数十万对象。

问题的本质在于 Paxos / Raft 等算法中，超过 99.99% 的代价都消耗在了网络广播（投票）和落盘等行为上。而这些行为的目的就是要保证数据的可靠性（数据要同时存储在多数节点的持久化设备上）。而服务发现、服务选举、故障检测、故障转移、故障恢复、分布式锁、任务调度等分布式协调功能所涉及到的恰恰又都是没有长期保存价值的临时性数据。因此花费超过 99.99% 的精力来持久化地保存它们的多个副本是毫无意义的——就算真的发生主节点下线等罕见灾难，我们也可以极高的效率，在瞬间就重新生成这些数据。

就好像张三买了一辆车，这辆车有个附加保险服务，其条款为：在张三万一发生了致命交通意外时，它能提供一种时光倒流机制，将其带回到意外发生之前的一瞬间来避免这场意外的发生。当然，这么牛的服务肯定也很贵，它大概需要预付张三家族在接下来的三生三世里能获得的所有财富。而且即使张三在驾驶这辆车的过程中，始终未发生过致命交通事故，那这些预先支付的服务费也是一分钱都不能减免的。这么昂贵的服务，且不说一般人一生中大概率都不会发生致命交通事故（更别提还要指定具体的某辆车）。即使真发生了，这个三代赤贫的代价也难说就值得吧？

而我们则为自己的汽车产品提供了另一种不同的附加服务：虽然没有时光倒流功能，但我们的服务可以在张三发生致命事故后，将所有受害方全体连车带人瞬间原地满血复活（是一根头发丝都不会少、一块漆皮都不会掉那种满血）。最关键的是，该服务无需预先收取任何费用。张三

只需要在每次这样的灾难发生以后，支付相当于其半个月的工资的再生技术服务费就可以了。

综上，我方专利的分布式协调算法，在提供与传统 Paxos/Raft 算法相同等级的强一致性和高可用性保证之同时，极大地降低了系统对网络和磁盘 IO 的依赖，并显著提升了系统整体性能和容量。对于大规模、强一致分布式集群的可用性（HAC）和性能（HPC）等指标均有显著提升。

关于 BYPSS 服务的进一步描述，详见：5.2.3 白杨消息端口交换服务（BYPSS）。

1.4.3 高效、高强度的密码编码学组件

包含公钥算法、对称加密算法、数据编解码、散列和消息验证算法、数据压缩算法等基础功能组件。除此之外，支撑平台还提供了多个经过高度抽象、可开箱即用的高级密码编码学功能组件，例如：

支持实时压缩和强加密的虚拟文件系统（VFS），VFS 支持包括 AES（128/256）、TwoFish 等在内的数十种强加密算法，使用 AES-NI、SSE4 等汇编指令集优化，效率高。在蓝鲸、白豚、职业精等全线产品中，我们均使用该组件为产品的数据库和配置类数据提供整库级的实时压缩和强加密保护。另外还包括基于公钥体系架构（PKI）的强加密通信保护组件等。

近年来安全问题频发，亚马逊、沃尔玛、Yahoo、Linkedin（领英）、索尼、摩根大通、OpenAI（ChatGPT）、UPS、eBay、京东、支付宝、一号店、协程、12306、网易、CSDN、中国人寿、以及各大酒店集团（如家、汉庭、锦江、洲际、喜来登、万豪等）等国内外知名企业均频频报出大量用户信息泄露的严重安全事件，安全保障已经刻不容缓。

我方所有数据库（整库）和本地配置数据均存放在我方自主研发的，支持实时（on-the-fly）数据压缩和强加密的虚拟文件系统（VFS）中进行全方位保护。支持数十种业界公认的强加密安全算法，即使系统管理员也无法窥视企业数据。

基于业界标准的强加密算法保证了即使未来出现了每秒钟可完成一千万亿次密钥破解尝试的超级计算机，也需要平均五千四百万亿年才能破解一个密钥。安全性得到了极大保证（具体可参考《白杨应用支撑平台技术白皮书》中的第 4 节）。

1.4.4 数据查询分析引擎

支撑平台中还包含了表达力优于 SQL 的查询分析引擎，自有查询引擎除了可以摆脱对特定 DBMS 的依赖，使我们的产品可以自由地在 MySQL、MS SQL Server、Oracle、DB2、SQLite 等 RDBMS 以及 MongoDB、Cassandra 等 NoSQL 数据库间灵活切换。更增加了基于 UNICODE 字符集的 ARE 高级正则查询、表中套表的关联查询、复杂虚拟字段等 SQL 没有的高级特性。

查询引擎通过汇编优化的 C/C++ 代码实现词法分析、语法分析、语义分析/中间代码生成、优化等步骤，并可在 2010 出厂的 Thinkpad W510（4 核 8 线程，主频 1.6G）笔记本上，仅用其中

一核一线程即可达到每秒 1300 万次以上的表达式求值效率。

1.4.5 更多…

我们并不依靠“商业秘密”来保护核心竞争力。相反，我们使用更公开透明的商标、认证、版权、专利和公证保管等手段来保护自己的合法权益。因此，我们的技术细节均公开于相应的文档中，详情可见《白杨应用支撑平台技术白皮书》：

http://baiy.cn/doc/asp_whitepaper.pdf
或者：http://baiy.cn/doc/asp_whitepaper_en.pdf（英文版）

等文档，包括 Hacker News（全球最大的计算机科学新闻网站）、Google Blogger、CSDN 以及博客园在内的多家国内外媒体均转载或报道了这些论文。相较于“严守秘密”，我们相信公开透明下的大量同行审评，加上实际生产环境下的严酷考验，更有利于产品品质的提升。

2. 系统总体架构

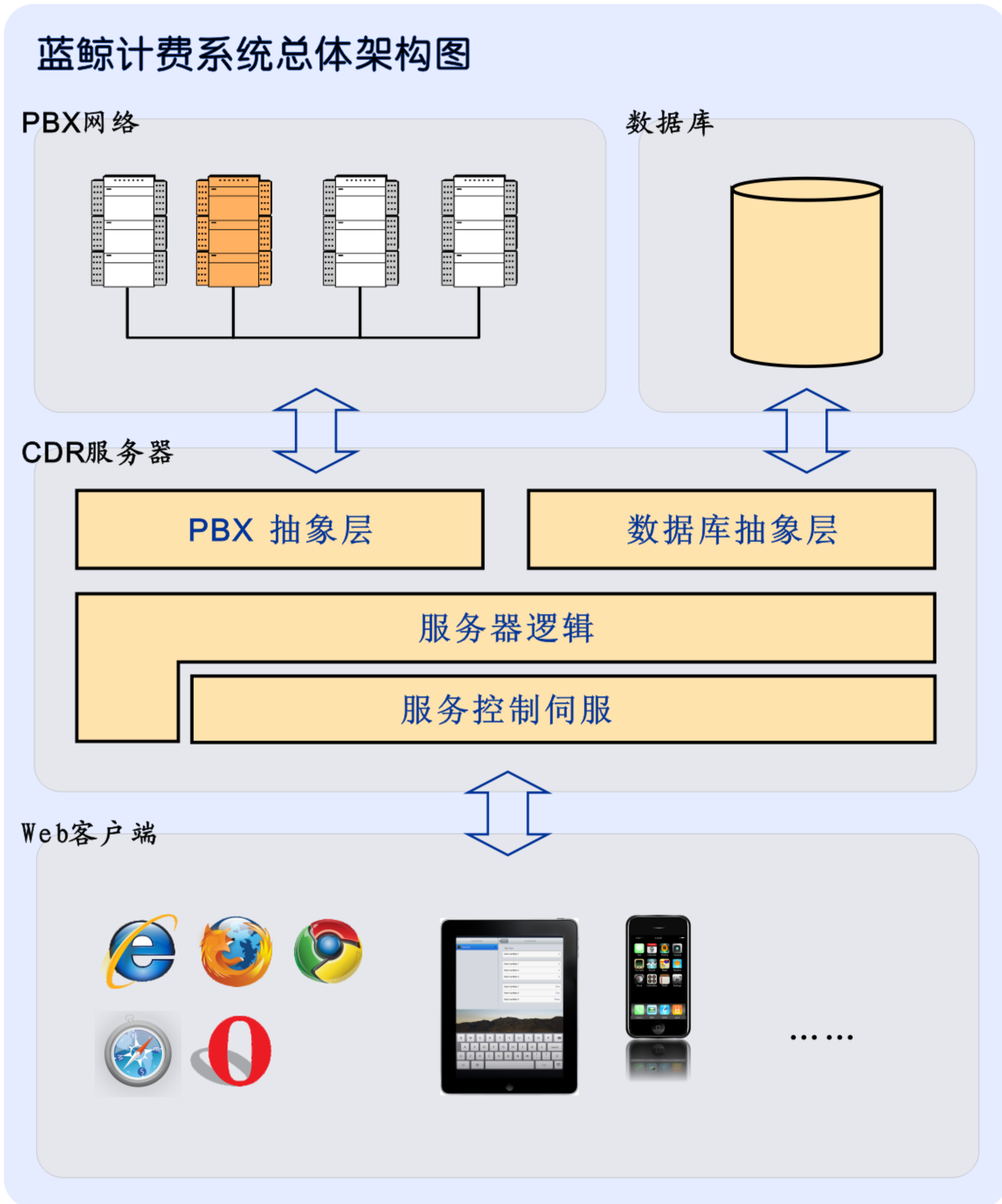


图 17

如图 17 所示，蓝鲸计费系统（以下简称"计费系统"）以成熟的数据库、业务逻辑（CDR 服务器）和 B/S 客户端三层构架实现。

2.1 数据库

数据库层作为数据存储中心，保存所有 CDR 话单以及与业务相关的各类数据，并提供事务（Transaction）、复制（Replication）、关系操作、视图、触发器等完成数据查询与管理的必要支持。为了做到透明支持不同种类的数据库系统，CDR 服务器将使用专门的抽象适配层（DBC）与数据库进行通讯。

2.2 PBX 网络

蓝鲸计费系统能够同时支持包含使用不同通讯协议（如：RSP、FTP 和串口等）、不同 CDR 格式、品牌各异的交换机混编网络。在开始计费之前，交换机和服务器都必须被正确地配置，并且相应类型的连接（IP 网络、串口连接等）都必须准备妥当。主要配置项目包括通信协议、通信地址和/或端口，以及话单格式等。

为了实现对不同通讯协议和 CDR 报文格式的透明支持，蓝鲸计费系统中的 CDR 服务器使用了一个专门的 PBX 通讯与话单解析抽象层（PBXC，PBX 连接器）来完成计费系统与交换机间的信息交互。PBXC 以插件的形式提供，用户可以根据自己的需要灵活选择一种或多种需要的 PBXC。

2.3 CDR 服务器

CDR 服务器位于客户端及数据库层之间，向上封装所有数据库操作和 PBX 通讯，同时采集、整理、分析、记录其管辖的所有 PBX 的话单信息；向下接受和处理由客户端发来的命令和请求；在双机容错模式中，还要横向与其伙伴服务器进行通讯。

CDR 服务器完成了所有面向业务的关键应用，包含复杂的业务处理逻辑和种类繁多的各种通讯接口及协议，是整个系统中的核心组件。

2.4 Web 客户端

蓝鲸计费系统提供基于 HTML5 技术的新一代 B/S 架构 Web 客户端，精心设计的新型客户端支持响应式布局的国际化多主题界面。可兼容包括 IE、Firefox、Chrome、Opera、Safari、iOS（iPhone/iPad）、Android 以及 BlackBerry 在内的各种浏览器和移动平台。可轻松实现跨平台的移动办公。

客户端是遵循功能与界面完全分离、以及 one page one application 的原则来进行设计的。这不但为用户提供了更快速的响应，以及更接近 C/S 架构的用户体验，开放式的 RESTful WebAPI

接口也使得二次开发工作变得简单易行。

3. 系统总体设计

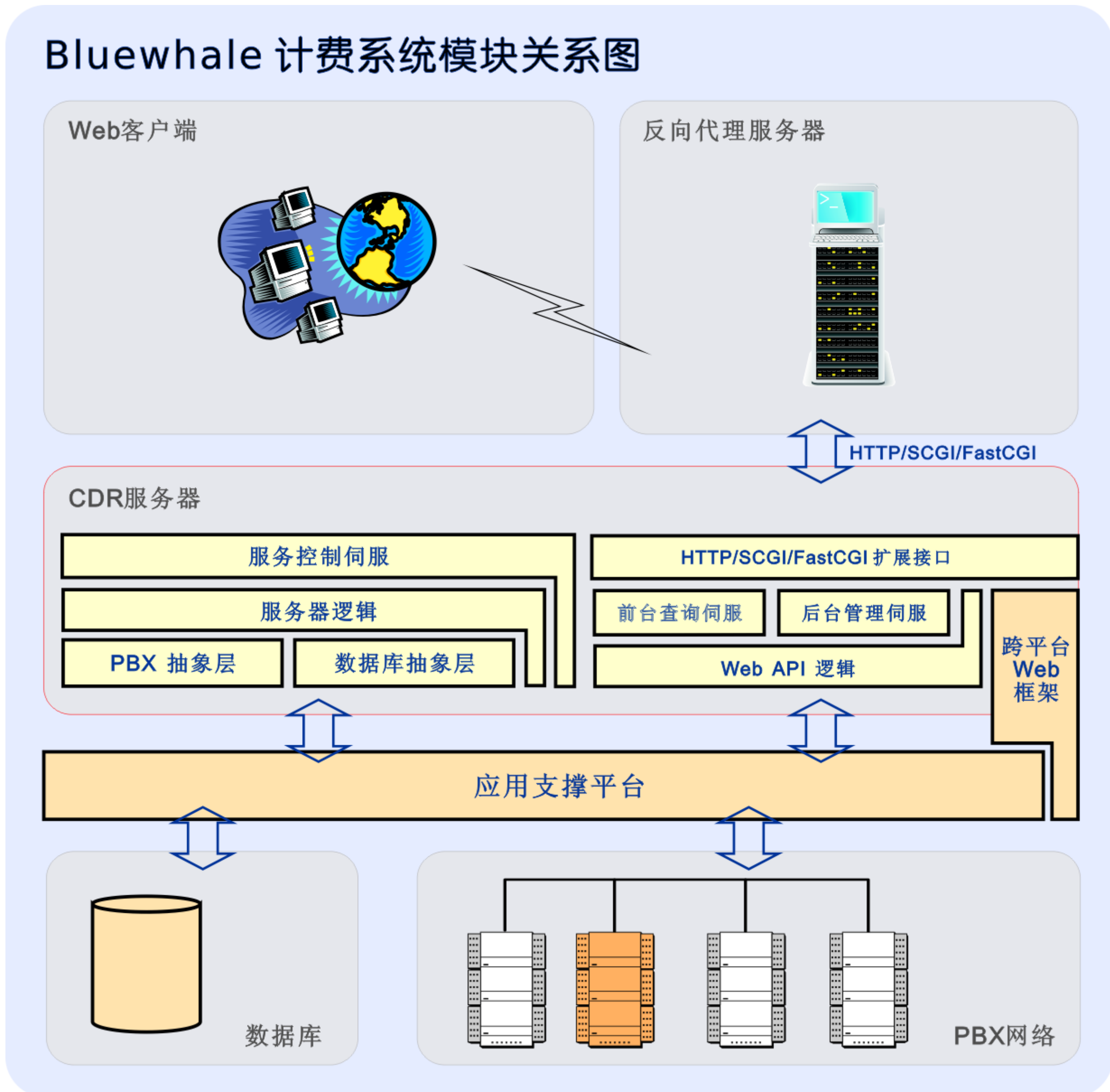


图 18

如图 18 所示，蓝鲸计费系统主要可分为应用支撑平台和 CDR 服务器两大层面，以下逐一讨论：

3.1 应用支撑平台

应用支撑平台是整个计费系统的基石，是蓝鲸计费与底层操作系统及硬件平台间的重要接口，应用支撑平台在完整封装操作系统功能的同时，还提供各种通用工具。应用支撑平台是高质

量、低成本快速开发及支持跨平台应用的关键组件。

3.1.1 应用支撑平台功能描述

应用支撑平台为计费系统提供了众多基础功能，包括：

* 跨平台底层支持：封装所有与操作系统相关的操作，如：信号量、原子操作、共享内存／文件映射、进程、线程、网络操作（Socket）、文件管理、服务控制、注册表访问、进程间通讯、服务器框架、异步 IO 框架等等。是计费系统实现跨平台、多平台的关键组件。

* 通用功能：包括用户权限管理、基于 PKI 体系结构的强加密算法、常用网络协议、二进制和字符集编码转换、自动化脚本引擎、表单处理、数据压缩、任务管理、日志记录、LRU Cache 容器等等。

* 平台无关的高效 Web 扩展框架。

* 平台无关的复杂数据结构表示以及支持实时压缩和强加密的虚拟文件系统。

* 平台无关的国际化支持：为用户提供一个平台无关的多语言、国际化工作环境。

等等。应用支撑平台完整封装了几乎所有与底层系统相关的功能，以及很多常用的算法、框架和功能组件。

3.1.2 应用支撑平台设计框架

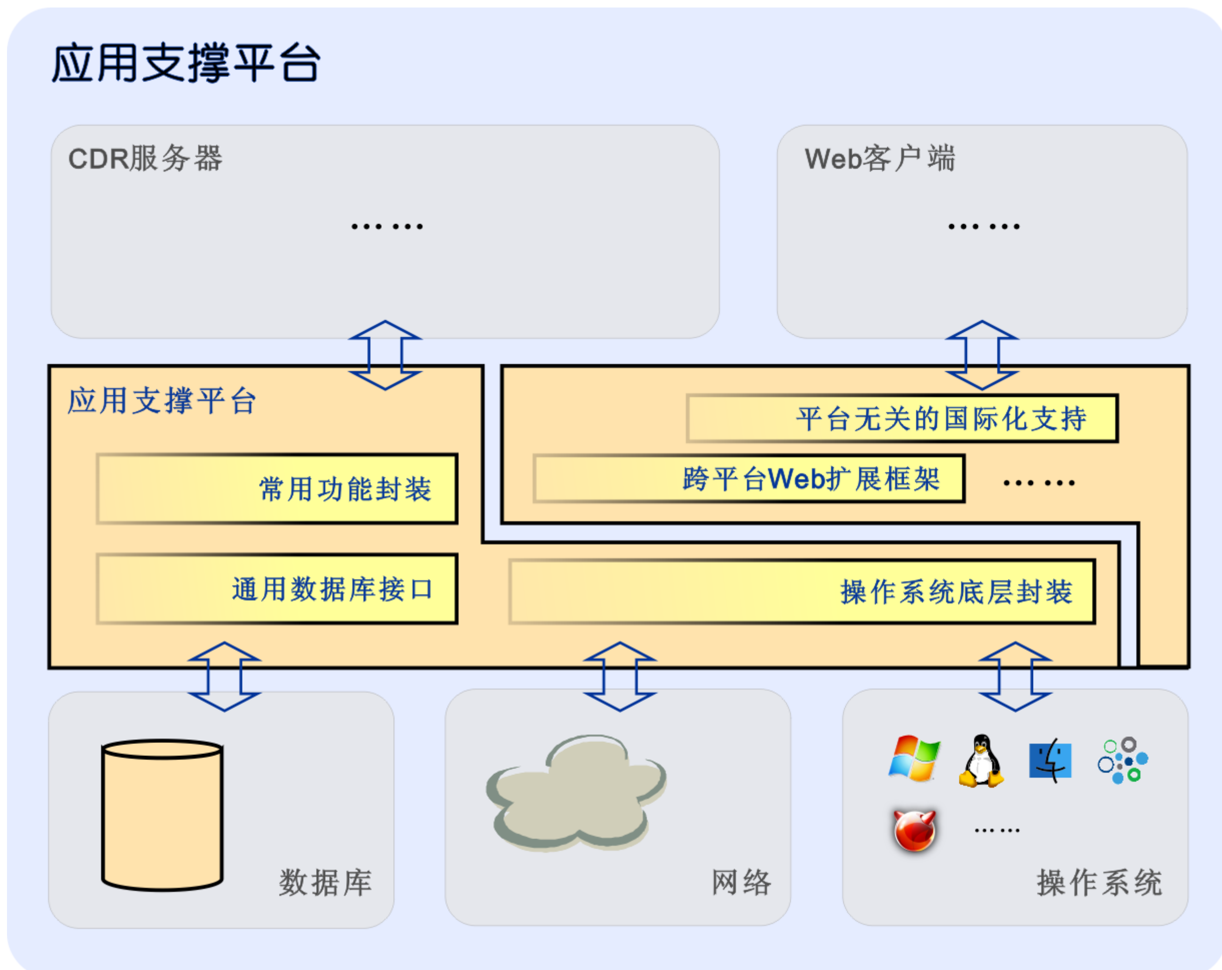


图 19

如图 19 所示，应用支撑平台位于操作系统、网络、数据库等底层环境及 CDR 服务器、Web 客户端等高层应用之间，是计费系统的构造基础。向下封装各类底层的、操作系统和运行环境相关的通用功能，向上为蓝鲸计费系统提供稳定强大、平台无关的通用服务和常用组件。

应用支撑平台为我方自主开发的跨平台应用支撑框架：跨平台框架统一封装了进程；线程；线程本地存储；信号量；共享内存；命名管道；文件映射；原子操作；消息机制；网络访问；文件系统管理；系统时钟；任务计划；服务控制；动态库装载等各类操作系统功能。

同时提供了基于 PKI 体系架构的高强度数据加密；基于数字证书的 License 生成和验证；数据压缩算法；通用句柄操作；数据编码转换；字体访问；字符集转换；数据库访问；报表处理；图像生成；用户鉴权和访问控制机制；音频、视频输入输出及编解码；嵌入式自动化脚本引擎等各类通用功能。

支撑平台从一开始就是专为对负载和可靠性要求很高的企业及互联网行业而设计的。其中的每个组件都会在保证可靠性的前提下，使用对当前底层平台来说，最高效的方式予以实现。

例如：支撑平台中的网络 IO 和 Web 扩展框架总是会尽可能地利用底层系统所支持的最优模式来完成网络访问：在 Windows 上使用 IOCP；在 Linux 上使用 epoll；在 FreeBSD/NetBSD/OpenBSD 等 BSD 环境中使用 kqueue；在 Solaris 上使用 event ports；在 HP-UX 上使用 /dev/poll、在 IBM AIX 上使用 pollset、而在其它 POSIX 环境下使用 POSIX AIO + Realtime Signal 等等。

再比如：根据实际情况直接使用内联汇编或编译器 Intrinsics 对原子量操作、内存屏障以及字节序交换等热点功能进行优化。并直接使用这些已被极度优化的组件构建快速互斥量、自旋锁、以及快速信号量等组件。

应用支撑平台完全使用高效强大的 C++ 语言和少量汇编代码进行开发，由一组跨平台的功能组件构成，全面支持当今流行的各种软、硬件环境[(1)]。目前已被广泛应用于多个大型项目[(2)]，其高效性、可靠性和成熟度已被广泛验证。

使用统一的应用支撑平台和用户界面组件构建软件可为用户带来很多显而易见的好处，例如：

- 更高的产品质量：大量重用经过岁月积累和时间考验的组件显著提高了产品的稳定性。与此同时，重用经过反复调优的组件也有助于提高性能。
- 跨平台能力和国际化支持：统一应用支撑平台和统一用户界面组件库封装了几乎所有系统相关的操作，并提供了很强的跨平台支持能力，以它们为基础开发的应用因此具有平台无关特性。典型情况下，跨平台移植的全部工作仅仅是对应用的简单重编译。同时，国际化作为统一用户界面组件的一个重要附加功能，为应用提供了平台无关的国际化支持能力。
- 极高的运行时效率：100％ 高效 C++/汇编编码，无论是时间效率还是空间效率上都没有任何妥协。

应用支撑平台包含繁多的功能和复杂的设计。其复杂度甚至还要远超蓝鲸计费系统本身。作为一个完善、独立的产品，应用支撑平台的具体细节已经超出了本文的讨论范围，关于和应用支撑平台相关的进一步信息，请参考：《应用支撑平台技术白皮书》（http://baiy.cn/doc/asp_whitepaper.pdf）。

注 1：我们当前支持的操作系统主要包括：

- Windows 98/ME, Windows NT4/2000/XP/2k3/Vista/2k8/Win7/2k8r2。
- Linux、FreeBSD、NetBSD、IBM AIX、HP-UX、Solaris、MAC OS X 系统及众多 Un*x/POSIX 系统。
- vxWorks, Nuclear, QNX, SMX, DOS, Windows CE (Windows Mobile), NanoGUI、eCos、RTEMS、Android、iOS 等嵌入式系统。

当前支持的主要硬件平台包括：x86/x64、ARM、IA64、MIPS、POWER、SPARC 等。

注 2：应用支撑平台的典型客户中，包括了兴业银行（China CIB）、中石油（CNPC）、华安保险（Sinosafe Insurance）、淘宝网（taobao.com）、烟台万华集团、法国兴业银行（SOCIETE GENERALE）、德尔福汽车（Delphi）、美联航（United Airlines）、GE（美国通用电气）、贝塔斯曼（Bertelsmann）、埃森哲（Accenture）等各大企业。

3.1.2.1 跨平台 Web 扩展框架

跨平台 Web 扩展框架作为应用支撑平台中的组成部分，实现了通用的 HTTP 伺服框架。应用支撑平台分别提供了同步和异步两种 Web 扩展框架。

由于需要支持高并发环境，前台 Web 伺服器框架使用高效的异步 IO 架构来实现，其工作模型如下图所示：

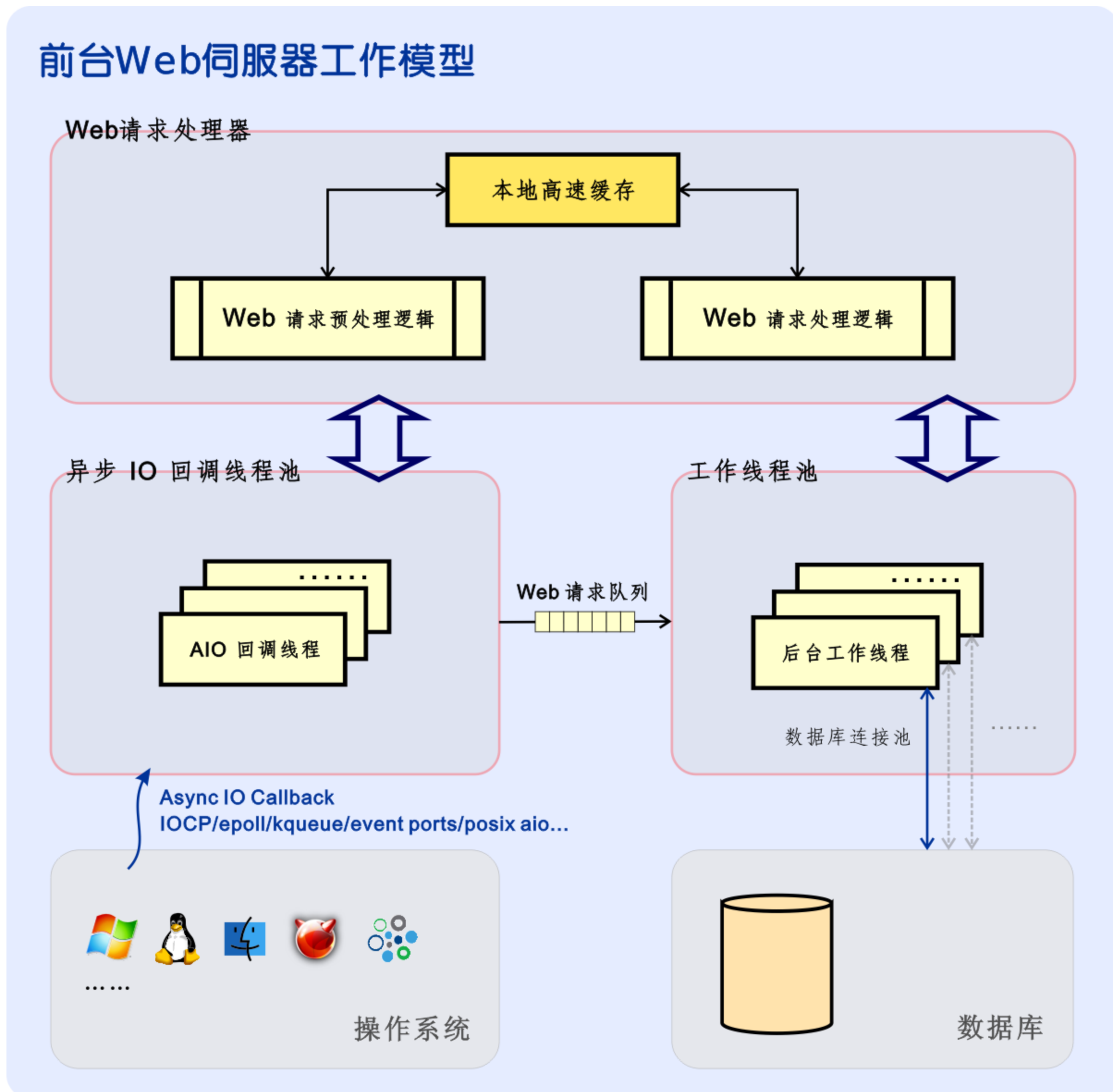


图 20

如图 20 所示，前台 Web 伺服器工作流程如下：

* 当一个 Web 请求到达后，底层操作系统通过 IOCP、epoll、kqueue、event ports、real time signal (posix aio)、/dev/poll、pollset 等各类与具体平台紧密相关的 IO 完成（或 IO 就绪）

回调机制通知异步 Web 扩展框架的 AIO 回调线程，对这个已到达的 Web 请求进行处理。

* 在 AIO 回调池中的工作线程接收到一个已到达的请求后，首先尝试通过 Web 请求处理器对该请求进行预处理。在预处理过程中，会先尝试根据 URI 匹配结果来回调相应的业务模块，若无匹配或业务模块放弃处理，则会继续尝试使用位于本地的高速缓存来避免成本较高的磁盘访问及数据库查询。如果业务模块的预处理请求成功，或本地缓存命中，则直接将缓存中的结果（仍然以异步 IO 的方式）返回客户端，并结束本次请求。

* 如果指定的 Web 请求要求查询的数据无法被本地缓存命中，或者这个请求需要数据库写入操作，则该请求将被 AIO 回调线程追加到指定的队列中，等待工作线程池中的某个空闲线程对其进行进一步处理。

* 工作线程池中的每个线程都分别维护着两条长连接：一条与底层到数据库服务相连，另一条则连接到分布式缓存（memcached）网络（可选，图中省略）。通过让每个工作线程维护属于自己的长连接，后台工作线程池实现了数据库和分布式缓存服务的长连接池机制。长连接（Keep-Alive）机制通过为不同的请求重复使用同一条网络连接大大提高了应用程序和网络 IO 的处理效率。

* 工作线程在 Web 请求队列上等待新的请求到达。在从队列中取出一个新的请求后，工作线程将该请求和可用的数据库连接提交给 Web 请求处理器进行处理。

* 请求处理器首先使用工作线程指定的数据库和分布式缓存服务长连接处理指定的 Web 请求（如果存在匹配的 URI 规则，则将其送交相应的业务模块进行处理）。然后将处理结果提交给底层 Web 扩展框架，将结果通过异步 IO 的方式返回给客户端。

应注意到：前台 Web 伺服器中的 Web 请求处理器及 Web 请求预处理均允许业务模块进行注册。所有已正确注册的业务模块都可以通过 URI 匹配等方式参与 Web 请求的处理和预处理机制。这就使得业务模块可以对 B/S 界面和 WebAPI 进行扩展，从而向用户暴露自己的外部接口。

需要进一步说明的是，与 epoll/kqueue/event ports 等相位触发的通知机制不同，对于 Windows IOCP 和 POSIX AIO Realtime Signal 这类边缘触发的 AIO 完成事件通知机制，为了避免操作系统底层 IO 完成队列（或实时信号队列）过长或溢出导致的内存缓冲区被长时间锁定在非分页内存池，在上述系统内的 AIO 回调方式实际上是由两个独立的线程池和一个 AIO 完成事件队列组成的：一个线程池专门负责不间断地等待系统 AIO 完成队列中到达的事件，并将其提交到一个内部的 AIO 完成队列中（该队列工作在用户模式，具有用户可控的弹性尺寸，并且不会锁定内存）；与此同时另一个线程池等待在这个内部 AIO 完成队列上，并且处理不断到达该队列的 AIO 完成事件。这样的设计降低了操作系统的工作负担，避免了在极端情况下可能出现的消息丢失和内存泄露，同时也可以帮助操作系统更好地使用和管理非分页内存池。

相反，由于不需要考虑高并发和高负载等使用场景，后台管理 Web 伺服器使用较简单的多线程，每连接一线程的阻塞 IO 模式：

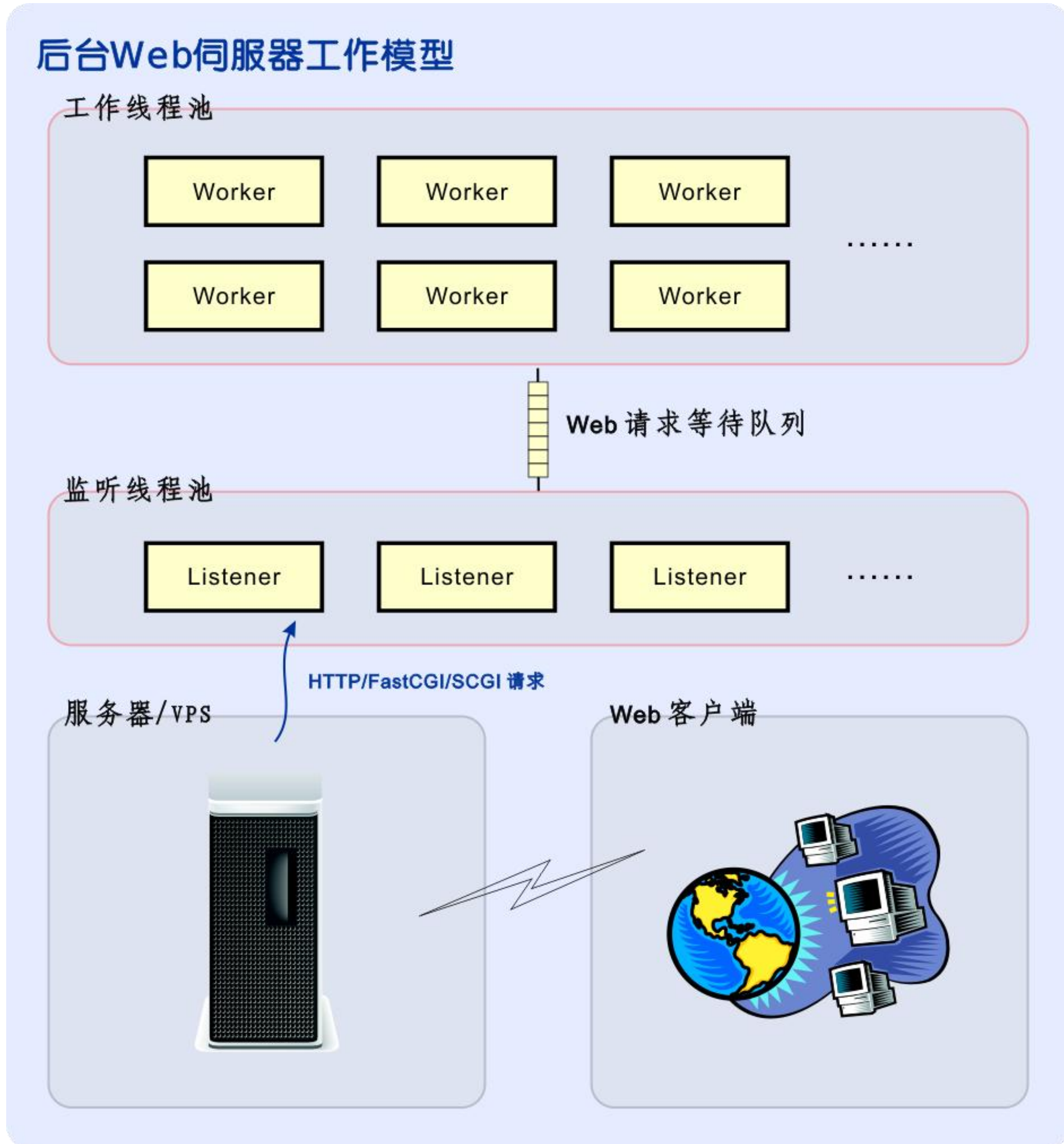


图 21

如图 21 所示，后台 Web 伺服器工作流程如下：

* 监听线程池等待到达用户和会话管理服务器的 Web 请求，并完成与该请求相关的初始化工作。然后将其压入 Web 请求队列的尾端，并继续监听下一个请求。为了增强高负载模式下的并发性，这些工作由单独的线程池来完成。监听线程池中的线程数量可以由用户配置，也可以根据负载情况同态调整。

* 工作线程池不断从等待队列的首部取出待处理的请求，将其提交给 Web 请求处理器。Web 请求处理器可视情况将其转交给相应的业务模块，最后返回处理结果。工作线程池可以根据当前负载状况和用户配置的参数进行动态调整。在处于轻载或空闲时，工作线

程的数量保持在用户指定的最小值。当负载持续加重时，为提供额外的并发处理能力，线程数量会逐渐增加，直到达到用户指定的最大值为止。当业务高峰结束，负载量持续下降时，应用框架会逐渐回收已空闲的工作线程，直到达到用户指定的最小值为止。工作线程的回收策略可以由用户指定。

3.2 CDR 服务器

CDR 服务器是蓝鲸计费系统中的关键功能组件，其向下接收和解析来自 PBX 的原始话单数据、完成话单解析和费率计算等工作，并负责将它们安全地写入数据库。向上接受来自 CDR 客户端的控制和查询请求，并返回执行结果。

3.2.1 CDR 服务器功能描述

从功能上说，CDR 服务器是蓝鲸计费系统中最为重要的组成部分。计费系统中的所有核心业务逻辑均在此处完成。CDR 服务器完成的主要功能包括：

- PBX 通讯服务：与指定 PBX 建立连接，并获取话单信息。
- CDR 解析服务：按照管理员定义的规则分析获取到的 CDR 记录，将其解析为统一内部表示。
- 数据库访问服务：通过数据库通讯抽象适配层将收集并解析到的 CDR 话单数据提交到指定数据库系统进行保存。
- 数据可靠性确保服务：当数据提交失败或无法连接到指定数据库时，使用可靠的，面向事务的机制，将数据库故障期间产生的所有话单统一保存到本地可靠存储（脱机数据库）中，待数据库恢复时重新提交。
- 数据库在线备份和恢复服务：在不中断服务的前提下，在线对所有话单和配置数据进行热备份和在线恢复。
- 用户鉴权和管理服务：根据安全等级需要进行基于用户名 / 密码或基于数字证书的用户鉴权，维护和管理用户/用户组的权限设置。
- 通讯加密服务：提供利用 PKI 体系结构，基于数字证书的强加密逻辑通讯链路。
- 客户端访问服务：实现完整的 RESTful Web API 接口，接受并处理来自客户端的管理和查询请求。
- 服务器协作服务：在异地容灾、双机容错方案中支持服务器簇内部的心跳信号、控制信令、数据复制等通讯和操作。

3.2.2 CDR 服务器设计框架

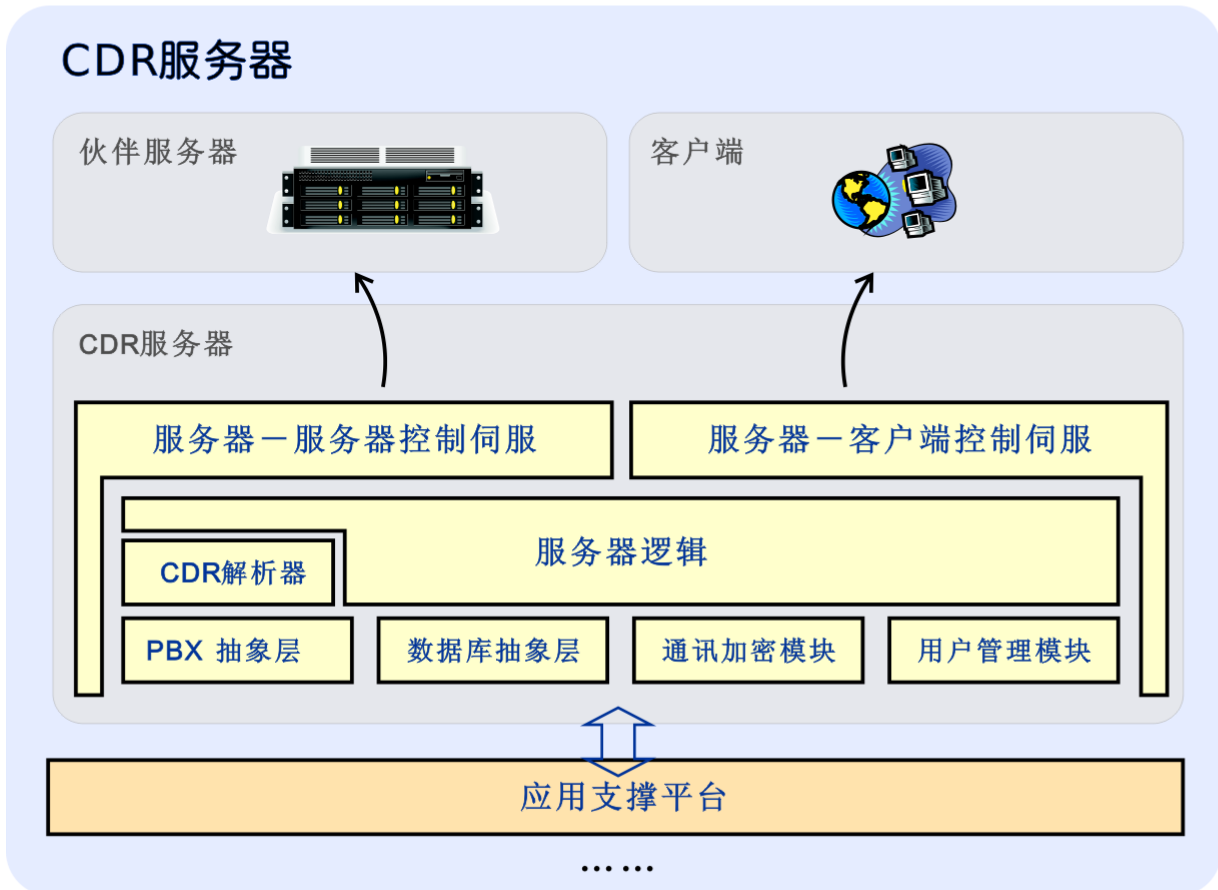


图 22

如图 22 所示，CDR 服务器基于应用支撑平台实现，实现了所有的业务逻辑。其向上连接客户端，向下封装数据库连接、操作和 PBX 通信、话单解析等功能。在开启了负载平衡和双机容错的应用中，CDR 服务器还要与其伙伴保持横向联系。

针对 Avaya Communication Manager，蓝鲸计费系统目前支持两种不同的 PBX 通信服务：一种是标准的、基于 TCP 的话单传输协议。该协议的优点是原理简单、实现简便，但其稳定性、可靠性和性能都明显劣于另一种基于 RSP 可靠协议的 PBX 通信服务。

可靠会话协议（Reliable Session Protocol，**RSP**）是由朗讯公司贝尔实验室定义，并广泛用于 AvayaCommunication Manager 的一套可靠会话层网络传输协议。该协议在 TCP/IP 基础上实现，作为 Avaya 主流交换产品中的高级功能提供给用户。该功能的主要目的是解决 TCP 协议无保证传输（TCP 仅提供尽力传输，在发生错误后，用户只能知道发生了错误，但无法得知数据丢失的具体情况）带来的不可靠性，为用户提供一套会话级的数据可靠交换及处理方式。同时，RSP 协议也实现了面向消息的会话层语义，Delay Buffer、传输窗口，以及会话控制和中断恢复等机制。

因此，使用 RSP 协议与 Avaya PBX 建立 CDR 传输连接，能够使得计费系统在性能（Delay Buffer）、可靠性和精确性等方面得到很大提高。RSP 也是 Avaya 官方建议使用的计费协议。

蓝鲸计费系统使用完全自主开发的 RSP 协议栈，全面支持协议标准中定义的各种可选项。在兴业银行、华安保险、中石油、TPC 外包型呼叫中心等客户生产环境中的多年实际运行证明，蓝鲸计费系统的 RSPPBXC 能够非常出色地与 Avaya 交换机长时间建立高效稳定的CDR 传输会话。

与此同时，蓝鲸计费系统也提供了针对 Cisco Call Manager 和 Call Manager Express 等产品的 PBX 通信服务。在这些产品中，我们同样使用了完全自主研发的，基于状态机和内存池缓存策略的专用话单接收与解析模块，从而保证了高效、稳定、并且可靠地完成了 CDR 话单的接收与解析。

CISCO、Nortel 等 PBX 产品在处理电话会议（Conference Call）、呼叫转接（Call Transfer）、呼叫前转（Call Forwarding）、cBarge、Tandem Call 及其各类组合情形时，会在话单流中为其分散地陆续产生多条相关话单。此时计费系统需要在长达数十小时时间窗口的数据流中，准确识别属于同一组通话活动的各个相关话单，并将他们拼凑在一起，最终还原出实际通话场景。

为处理此类情形，蓝鲸计费系统引入了自主开发的复杂事件处理（Complex Event Processing，CEP）引擎，确保其能够准确、高效地分析和处理在海量数据中离散分布的各类关联话单。并能够自动跟踪、记录合并和处理各话单之间的不同逻辑关系以及这些逻辑关系出现各种复杂组合的情况，确保最终的计费结果精准可靠。

此外，针对高负载应用和对可靠性要求很高的场合。蓝鲸计费也提供了相应的高性能集群（HPC），以及抗脑裂的强一致多活 IDC 分布式集群（HAC）解决方案。以下是 CDR 服务器 HAC+HPC 分布式集群方案示意图：

图 23

关于其中 BYPSS 分布式协调服务的进一步说明，详见：5.2.3 白杨消息端口交换服务（BYPSS）。

在上述强一致、抗脑裂的 HAC+HPC 方案中，企业内所有蓝鲸服务器被划分为若干组，每个服务器组内可包含一台或多台 CDR Server，集群可以服务器组为单位，被分别部署于不同的数据中心（机房），以实现多活 IDC 架构，防止单机房故障导致的服务不可用。企业中的每组 PBX 均同时与服务器簇中的多组 CDR Server 建立连接。

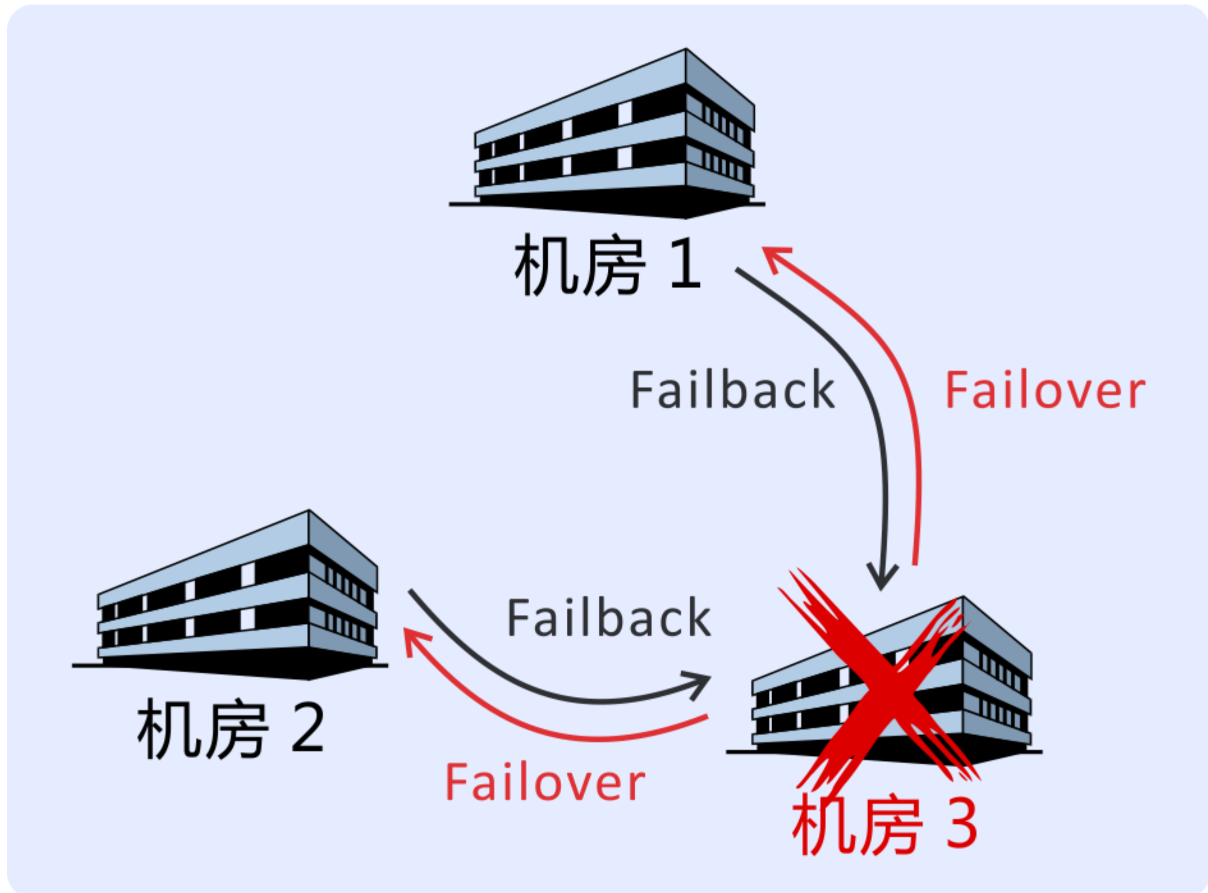


图 24

正常情况下，每个服务器组各自负责一组 PBX，忽略其它 PBX 递交的话单（只对上次数据库复制后的话单数据进行缓冲）。当 BYPSS 服务监测到某服务器节点或某机房发生故障宕机时，将通过合理的负载平衡算法，分配其它服务器组临时接管该 PBX 的计费工作（故障转移）。在检测到对应的服务器节点恢复后，集群将按照 BYPSS 的指示完成话单数据更新，并恢复正常工作模式（故障恢复），在确保极高可用性的同时合理平衡负载，维持集群整体的最高利用率。

3.2.2.1 话单处理流程

蓝鲸计费系统 CDR 服务器的话单接收、解析、批价与入库流程协作图如下所示：

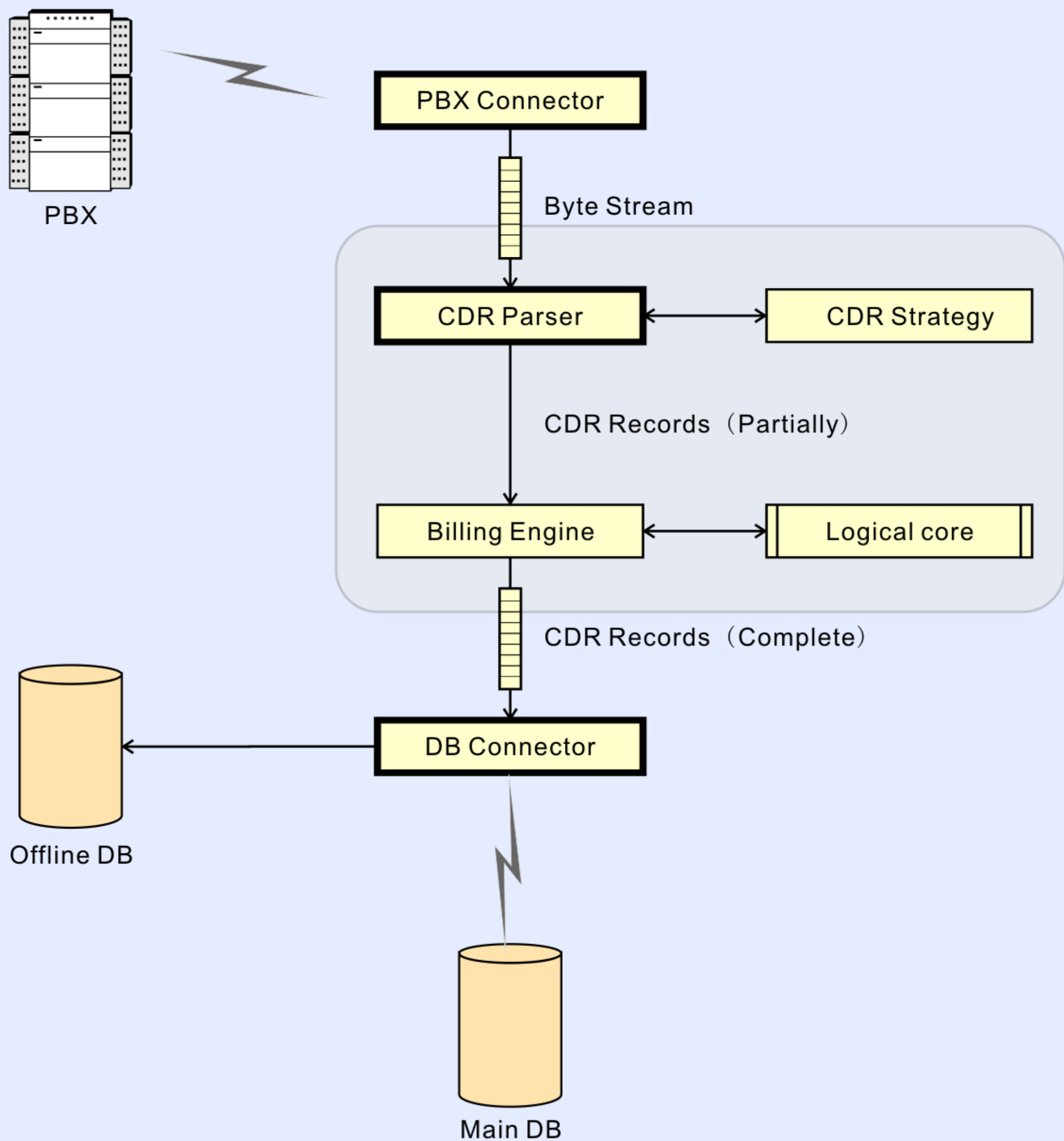


图 25

上图描述了从原始话单接收、话费计算到入库的完整流程，以及相关对象间的协作关系。其中 PBX Connetcor（PBX 连接器）负责从 PBX 接收话单，而中间由浅灰色圆角矩形涵盖的四个部分完成了话单解析与费率计算等工作，最后由 DB Connector（数据库连接器）将计算结果写入数据库并通过脱机数据库机制确保数据安全。

在整个话单处理流程中，话单接收；话单解析与计费和话单入库等三个关键模块均分别工作在自己的线程池中。它们之间使用多条由生产者/消费者模型同步的消息队列相互连接。这项技术被称作**“多级内存话单解析”**。该技术一方面保证了接收话单，数据入库等关键任务不会被其它任务干扰和阻塞，同时也能够有效地吸收话单浪涌。从而保证计费系统稳定运行，避免出现话单

丢失等问题。

4．远程单元

远程单元（Remote Unit，RU）实际上是一个简化版的 CDR 服务器，它以 Daemon/Service 的形式作为独立的进程运行在目标平台上。RU 通常被部署在离 PBX 很近，但远离蓝鲸 CDR 服务器的物理位置：

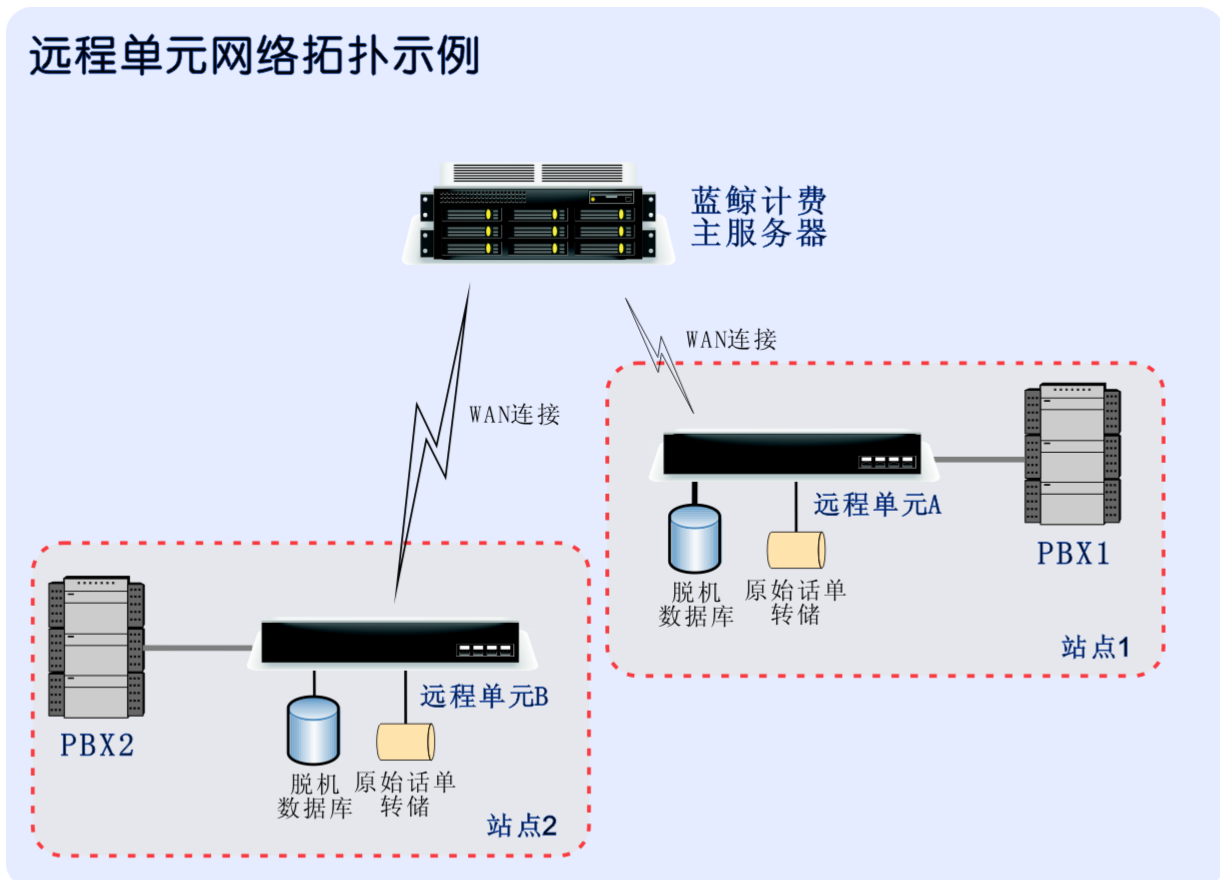


图 26

如图 26 所示，远程单元位于 PBX 和 CDR 服务器中间。向下连接 PBX，向上连接主服务器（CDR 服务器），并提供以下增强服务：

* **可靠性保证：**通过使用自身维护的 Offline DB 保证了即使主服务器与 RU 间的连接中断也不会造成消息丢失。此外，远程单元与主服务器之间通过可靠的会话层协议（RSP）进行通信，即使在恶劣的网络环境中，也能够有效避免话单丢失或重复等情况的发生。

* **安全性保证：**使用经过验证的安全鉴权模式和实时强加密算法提供消息的真实性、完整性和保密性保证。

* **节省带宽：**实时压缩算法保证了最小带宽占用，提高传输效率和节省昂贵的带宽资源。

* **原始话单转储**：对从易失性（如：COM 口、TCP 连接等）数据源取得的话单数据提供原始话单转储服务（管理员可配置仅保留最近多少天、以及每隔多长时间生成一个新的转储文件等参数）。转储话单为管理员提供了第一手资料和证据，并且允许通过转储文件重新导入话单。

* **高可用性保证**：远程单元可支持基于共享存储（如：Windows Cluster）或复制（如：Linux DRBD）的多点容错方案。

由于距离（网络节点跨度）PBX 更近、以及能运行在成本更低的嵌入式系统等原因，RU 通常能够以比主服务器更低的成本提供相同，甚至更好的高可用性保证，避免单点故障造成话单丢失。

5. 技术特点

本节集中讨论蓝鲸成本管理系统的主要技术特点和关键设计考量。

5.1 REST 风格的 Web API 接口

在综合考虑性能、网关穿透性、安全性、易用性和学习曲线等多方面因素后，蓝鲸成本管理系统使用 RESTful 风格的 HTTP 接口（也常被称为 Web API）为其它子系统提供功能调用。并通过 SSL/TLS 层为其提供通信安全保护服务。HTTP/HTTPS 作为当前应用最广泛的网络协议，已被几乎所有开发人员认可并熟知，并已有大量成熟的封装组件实现了完整的 HTTP 协议。这些因素使得其学习成本和易用性都非常理想。另一方面，HTTP/1.1 对 KeepAlive 和 Pipelining、以及 HTTP/2 对全双工多路复用通信的支持使其能够很好地工作在高并发、高负载环境。

在今天的网络环境中，Web API 已被广泛应用于当今各大站点中，例如：amazon.com 的 EC2/S3、淘宝的 TOP（即：淘宝提供给其各个 ISV 的开发接口）、以及 Google、Facebook、Twitter、Flicker、百度、腾讯等网站都以 Web API 的形式开放了自己的二次开发接口。

Web API 的优越性主要体现在以下几方面：

* **实现方便**：B/S 架构的应用本身就使用 HTTP 实现，只需要进行较少的扩展即可作为 API 提供给用户使用。

* **连通性好**：能够通过任何 HTTP Proxy 透传。

* **基础设施完备**：可基于大量现有的，经过高度优化和长时间测试的 Web Server、Web 扩展框架和反向代理来实现。可直接利用 HTTP 协议中已实现的 SSL/TLS 加密和 gzip/zlib 压缩算法完成数据传输的加密和压缩等功能。

* **使用 HTTP 语义**：不同于 SOAP 等仅将 HTTP 作为传输层协议的技术，RESTful HTTP 接口要求使用 HTTP 语义来进行通信，意即：请求中的 GET/PUT/POST/DELETE/PUT 等方法、URI 以及 HTTP 消息头和返回中的状态码及消息头均遵循标准的 HTTP 规范而代表着对应的行为和参数等含义，以能够真实反应调用的实际语义为佳。

* **无状态**：HTTP 本身为无状态协议，便于实现负载均衡、容错和提高实现效率。

5.2 nano-SOA 架构

5.2.1 SOA vs. AIO

长久以来，服务器端的高层架构大体被区分为对立的两类：SOA（Service-oriented architecture）以及 AIO（All in one）。SOA 将一个完整的应用分割为相互独立的服务，每个服务提供一个单一标准功能（如：会话管理、交易评价、用户积分等等）。服务间通过 RPC、WebAPI 等 IPC 机制暴露功能接口，并以此相互通信，最终组合成一个完整的应用。

而 AIO 则相反，它将一个应用规约在一个独立的整体中，SOA 中的不同服务在 AIO 架构下呈现为不同的功能组件和模块。AIO 应用的所有组件通常都运行在一个地址空间（同一进程）内，所有组件的代码也常常放在同一个产品项目中一起维护。

AIO 的优势是部署简单，不需要分别部署多个服务，并为每个服务实现一套高可用集群。与此同时，由于可避免网络传输、内存拷贝等 IPC 通信所带来的大量开销，因此 AIO 架构的单点效率通常远高于 SOA。

另一方面，由于 AIO 架构中组件依赖性强，组件间经常知晓并相互依赖对方的实现细节，因此组件的可重用性及可替换性差，维护和扩展也较困难。特别是对于刚加入团队的新人来说，面对包含了大量互相深度耦合之组件和模块的“巨型项目”，常常需要花费大量努力、经历很多挫折并且犯很多错误才能真正接手。而即使对于老手来说，由于模块间各自对对方实现细节错综复杂的依赖关系，也容易发生在修改了一个模块的功能后，莫名奇妙地影响到其它看起来毫不相干功能的情况。

与此相反，SOA 模型部署和配置复杂——现实中，一个大型应用常常被拆分为数百个相互独立的服务，《程序员》期刊中的一份公开发表的论文显示，某个国内“彻底拥抱”SOA 的著名（中国排名前 5）电商网站将他们的 Web 应用拆分成了一千多个服务。可以想象，在多活数据中心的高可用环境内部署成百上千个服务器集群，并且配置他们彼此间的协作关系是多大的工作量。最近的协程(ctrip.com)网络瘫痪事件也是因为上千个服务组成的庞大 SOA 架构导致故障恢复缓慢。

除了部署复杂以外，SOA 的另一个主要缺点就是低效——从逻辑流的角度看，几乎每次来自客户端的完整请求都需要依次流经多个服务后，才能产生最终结果并返回用户端。而请求（通过消息中间件）每“流经”一个服务都需要伴随多次网络 IO 和磁盘访问，多个请求可累计产生较高的网络时延，使用户请求的响应时间变得不可确定，用户体验变差，并额外消耗大量资源。

此外，无论是每个 Service 各自连接不同的 DBMS 还是它们分别接入同一个后端分布式 DBMS 系统，实现跨服务的分布式事务支持工作都要落到应用层开发者手中。而分布式事务（XA）本身的实现复杂度恐怕就已超过大部分普通应用了，更何况还需要为分布式事务加上高可靠和高可用保证——需要在单个数据切片上使用 Paxos/Raft 或主从+Arbiter 之类的高可用、强一直性算法，同时在涉及多个数据切片的事务上使用 2PC/3PC 等算法来保证事务的原子性。因此 SOA 应用中的跨 Service 事务基本都只能退而求其次，做到最终一致性保证，即便如此，也需要增加大量的

额外工作——在稍微复杂点的系统里，高可用，并能在指定时间内可靠收敛的最终一致性算法实现起来也不是那么容易。

与此同时，大部分 SOA 系统还经常需要使用消息中间件来实现消息分发服务。如果对消息中间件的可用性（部分节点故障不会影响正常使用）、可靠性（即使在部分节点故障时，也确保消息不丢失、不重复、并严格有序）、功能性（如：发布/订阅模型、基于轮转的任务分发等）等方面有所要求的话，那么消息中间件本身也容易成为系统的瓶颈。

SOA 架构的优点在于其高内聚、低耦合的天然特性。仅通过事先约定的 IPC 接口对外提供服务，再配合服务间隔离（通常是在独立节点中）运行的特质，SOA 架构划分出了清晰的接口和功能边界，因此可以被非常容易地重用和替换（任何实现了兼容 IPC 接口的新服务都可替换已有的老服务）。

从软件工程和项目管理的视角来看，由于每个服务本身通常有足够高的内聚性，并且单个服务实现的功能也较独立，因此相对于 AIO 意大利面式的，相互交织的结构来说，SOA 的服务非常便于维护——负责某一服务的开发人员只需要看好自己这一亩三分地即可，只要保持服务对外提供的 API 没有发生不兼容的变化，就不需要担心修改代码、替换组件等工作会影响到其它“消费者”。

同时，由多个独立服务所组成的应用也更容易通过加入新服务和重新组合现有服务来进行功能变更和扩展。

5.2.2 nano-SOA 架构

在经历了大量实际项目中的权衡、思索和实践后，我们逐步定义、实现和完善了能够兼两者之长的“nano-SOA”架构。并将功能插件（IPlugin）、数据库连接器（DBC）API Nexus 等关键性的基础功能组件封装在了《白杨应用支撑平台》中，以提供便于重用的高品质基础架构。

在 nano-SOA 架构中，独立运行的服务被替换成了支持动态插拔的跨平台功能插件（IPlugin）；而插件则通过（并仅可通过）API Nexus 来动态地暴露（注册）和隐藏（注销）自身所提供的功能接口，同时也使用 API Nexus 来消费其它插件提供服务。

nano-SOA 完全继承了 SOA 架构高内聚、低耦合的优点，每个插件如独立的服务一样，有清晰的接口和边界，可容易地被替换和重用。在可维护性上，nano-SOA 也与 SOA 完全一致，每个插件都可以被单独地开发和维护，开发人员只需要管好自己维护的功能插件即可。通过加入新插件以及对现有功能插件的重新组合，甚至可比 SOA 模式更容易地对现有功能进行变更和扩展。

而在性能方面，由于所有功能插件都运行在同一个进程内，因此通过 API Nexus 的相互调用不需要任何网络 IO、磁盘访问和内存拷贝，也没有任何形式的其它 IPC 开销，因此其性能和效率均可与 AIO 架构保持在相同量级。

与此同时，nano-SOA 的部署与 AIO 同样简单——部署在单个节点即可使用，只需部署一个

集群即可实现高可用和横向扩展。在配置方面也远比 SOA 简单，仅需要比 AIO 应用多配置一个待加载模块列表而已，并且这些配置也可通过各种配置管理产品来实现批量维护。简单的部署和配置过程不但简化了运营和维护工作，也大大方便了开发和测试环境的构建。

此外，nano-SOA 也在极大程度上避免了对消息中间件的依赖，取而代之的是通过 API Nexus 的直接 API 调用；或是在需要削峰填谷的场合中，使用由内存零拷贝和无锁算法高度优化的线程间消息队列。这一方面大大增加了吞吐，避免了延迟，另一方面也避免了部署和维护一个高可用的消息分发服务集群所带来的巨大工作量——nano-SOA 集群内的节点间协作和协调通信需求已被将至最低，对消息分发的可靠性、可用性和功能性都没有太高要求。在多数情况下，使用 Gossip Protocol 等去中心化的 P2P 协议即足以满足需要，有时甚至可以完全避免这种集群内的节点间通信。

从 nano-SOA 的角度看，也可以将 DBC 视作一种几乎所有服务器端应用都需要使用的基础功能插件，由于其常用性，因此他们被事先实现并加进了 libapidbc 中。由此，通过提供 IPlugin、API Nexus 以及 DBC 等几个关键组件，libapidbc 为 nano-SOA 架构奠定了良好的基础设施。

当然，nano-SOA 与 SOA 和 AIO 三者间并不是互斥的选择。在实际应用场景中，可以通过三者间的有机组合来达成最合理的设计。例如：对于视频转码等非常耗时并且不需要同步等待其完成并返回结果的异步操作来说，由于其绝大部分开销都耗费在了视频编解码计算上，因此将其作为插件加入其它 App Server 就完全没有必要，将它作为独立的服务，部署在配置了专用加速硬件的服务器集群上应该是更好的选择。

5.2.3 白杨消息端口交换服务（BYPSS）

白杨消息端口交换服务（BYPSS）设计用于单点支撑万亿量级端口、百万量级节点规模，每秒处理千万至十亿量级消息的高可用、强一致、高性能分布式协调和消息交换服务。其中关键概念包括：

* 连接（Connection）：每个客户端（应用集群中的服务器）节点至少与端口交换服务保持一个 TCP 长连接。

* 端口（Port）：每个连接上可以注册任意多个消息端口，消息端口由一个 UTF-8 字符串描述，必须在全局范围内唯一，若其它客户端节点已注册了相同的消息端口，则端口注册失败。

端口交换服务对外提供的 API 原语包括：

* 等待消息（WaitMsg）：客户端集群中的每个节点均应保持一个到端口交换服务的 TCP 长连接，并调用此方法等待消息推送。此方法将当前客户端连接由消息发送连接升级为消息接收连接。

  每个节点号只能对应一个消息接收连接，若一个节点尝试同时发起两个消息接收连接，

则较早的那个接收连接将被关闭，并且绑定到当前节点上的所有端口都将被注销。

* 续租（Relet）：若端口交换服务在指定的间隔内未收到来自某个消息接收连接的续租请求，则判定该节点已经下线，并释放所有属于该节点的端口。续租操作用来周期性地向端口交换服务提供心跳信号。

* 注册端口（RegPort）：连接成功建立后，客户端应向端口交换服务注册所有属于当前节点的消息端口。可以在一个端口注册请求中包含任意多个待注册端口，端口交换服务会返回所有注册失败（已被占用）的端口列表。调用者可以选择是否需要为注册失败的端口订阅端口注销通知。

  需要注意的是，每次调用 WaitMsg 重建消息接收连接后，都需要重新注册当前节点上的所有端口。

* 注销端口（UnRegPort）：注销数据当前节点的端口，可一次提交多个端口执行批量注销。

* 消息发送（SendMsg）：向指定的端口发送消息（BLOB），消息格式对交换服务透明。若指定的端口为空串，则向端口交换服务上的所有节点广播此消息；亦可同时指定多个接收端口，实现消息组播。若指定的端口不存在，则安静地丢弃该消息。客户端可在一次请求中包含多个消息发送命令，主动执行批量发送，服务器端也会将发往同一节点的消息自动打包，实现消息批量推送。

* 端口查询（QueryPort）：查询当前占用着指定端口的节点号，及其 IP 地址。此操作主要用于实现带故障检测的服务发现，消息投递时已自动执行了相应操作，故无需使用此方法。可在同一请求中包含多个端口查询命令，执行批量查询。

* 节点查询（QueryNode）：查询指定节点的 IP 地址等信息。此操作主要用于实现带故障检测的节点解析。可以一次提交多个节点，实现批量查询。

端口交换服务的客户端连接分为以下两类：

* **消息接收连接（1:1）**：接收连接使用 WaitMsg 方法完成节点注册并等待消息推送，同时通过 Relet 接口保持属于该节点的所有端口被持续占用。客户端集群中的每个节点应当并且仅能够保持一个消息接收连接。此连接为长连接，由于连接中断重连后需要重新进行服务选举（注册端口），因此应尽可能一直保持该连接有效并及时完成续租。

* **消息发送连接（1:N）**：所有未使用 WaitMsg API 升级的客户端连接均被视为发送连接，发送连接无需通过 Relet 保持心跳，仅使用 RegPort、UnRegPort、SendMsg 以及 QueryPort 等原语完成非推送类的客户端请求。客户端集群中的每个节点通常都会维护一个消息发送连接池，以方便各工作线程高效地与端口转发服务保持通信。

与传统的分布式协调服务以及消息中间件产品相比，端口转发服务主要有以下特点：

* 功能性：端口转发服务将标准的消息路由功能集成到了服务选举（注册端口）、服务发现（发送消息和查询端口信息）、故障检测（续租超时）以及分布式锁（端口注册和注销通知）等分布式协调服务中。是带有分布式协调能力的高性能消息转发服务。通过QueryPort 等接口，也可以将其单纯地当作带故障检测的服务选举和发现服务来使用。

* 高并发、高性能：由 C/C++/汇编实现；为每个连接维护一个消息缓冲队列，将所有端口定义及待转发消息均保存在内存中（Full in-memory）；主从节点间无任何数据复制和状态同步开销；信息的发送和接收均使用纯异步 IO 实现，因而可提供高并发和高吞吐的消息转发性能。

* 可伸缩性：在单点性能遭遇瓶颈后，可通过级联上级端口交换服务来进行扩展（类似 IDC 接入、汇聚、核心等多层交换体系）。

* 可用性：最低 5 毫秒内完成故障检测和主备切换的高可用保证，基于多数派的选举算法，避免由网络分区引起的脑裂问题。

* 一致性：保证任意给定时间内，最多只有一个客户端节点可持有某一特定端口。不可能出现多个客户端节点同时成功注册和持有相同端口的情况。

* 可靠性：所有发往未注册（不存在、已注销或已过期）端口的消息都将被安静地丢弃。系统保证所有发往已注册端口消息有序且不重复，但在极端情况下，可能发生消息丢失：

  - 端口交换服务宕机引起主从切换：此时所有在消息队列中排队的待转发消息均会丢失；所有已注册的客户端节点均需要重新注册；所有已注册的端口（服务和锁）均需要重新进行选举/获取（注册）。

  - 客户端节点接收连接断开重连：消息接收连接断开或重连后，所有该客户端节点之前注册的端口均会失效并需重新注册。在接收连接断开到重连的时间窗口内，所有发往之前与该客户端节点绑定的，且尚未被其它节点重新注册的端口之消息均被丢弃。

可见，白杨消息端口转发服务本身是一个集成了故障检测、服务选举、服务发现和分布式锁等分布式协调功能的消息路由服务。它通过牺牲极端条件下的可靠性，在保证了强一致、高可用、可伸缩（横向扩展）的前提下，实现了极高的性能和并发能力。

可以认为消息端口交换服务就是为 nano-SOA 架构量身定做的集群协调和消息分发服务。nano-SOA 的主要改进即：将在 SOA 中，每个用户请求均需要牵扯网络中的多个服务节点参与处理的模型改进为大部分用户请求仅需要同一个进程空间内的不同 BMOD 参与处理。

这样的改进除了便于部署和维护，以及大大降低请求处理延迟外，还有两个主要的优点：

* 将 SOA 中，需要多个服务节点参与的分布式事务或分布式最终一致性问题简化成为了本地 ACID Transaction 问题（从应用视角来看是如此，对于分布式 DBS 来说，以 DB 视角看来，事务仍然可以是分布式的），这不仅极大地简化了分布式应用的复杂度，增强

了分布式应用的一致性，也大大减少了节点间通信（由服务间的 IPC 通信变成了进程内的指针传递），提高了分布式应用的整体效率。

* 全对等节点不仅便于部署和维护，还大大简化了分布式协作算法。同时由于对一致性要求较高的任务都已在同一个进程空间内完成，因此节点间通信不但大大减少，而且对消息中间件的可靠性也不再有过高的要求（通常消息丢失引起的不一致可简单地通过缓存超时或手动刷新来解决，可确保可靠收敛的最终一致性）。

在此前提下，消息端口交换服务以允许在极端情况下丢失少量未来得及转发的消息为代价，来避免磁盘写入、主从复制等低效模式，以提供极高效率。这对 nano-SOA 来说是一种非常合理的选择。

5.2.3.1 极端条件下的可靠性

传统的分布式协调服务通常使用 Paxos 或 Raft 之类基于多数派的强一致分布式算法实现，主要负责为应用提供一个高可用、强一致的分布式元数据 KV 访问服务。并以此为基础，提供分布式锁、消息分发、配置共享、角色选举、服务发现、故障检测等分布式协调服务。常见的分布式协调服务实现包括 Google Chubby（Paxos）、Apache ZooKeeper（Fast Paxos）、etcd（Raft）、Consul（Raft+Gossip）等。

Paxos、Raft 等分布式一致性算法的最大问题在于其极低的访问性能和极高的网络开销：对这些服务的每次访问，无论读写，都会产生至少三次网络广播——以投票的方式确定本次访问经过多数派确认（读也需要如此，因为主节点需要确认本次操作发生时，自己仍拥有多数票支持，仍是集群的合法主节点）。

在实践中，虽可通过降低系统整体一致性或加入租期机制来优化读操作的效率，但其总体性能仍十分低下，并且对网络 IO 有很高的冲击：Google、Facebook、Twitter 等公司的历次重大事故中，很多都是由于发生网络分区或人为配置错误导致 Paxos、Raft 等算法疯狂广播消息，致使整个网络陷入广播风暴而瘫痪。

此外，由于 Paxos、Raft 等分布式一致性算法对网络 IO 的吞吐和延迟等方面均有较高要求，而连接多座数据中心机房（IDC）的互联网络通常又很难满足这些要求，因此导致依赖分布式协调算法的强一致（抗脑裂）多活 IDC 高可用集群架构难以以合理成本实现。作为实例：2015 年 5 月支付宝和 2013 年 7 月微信平台分别长时间下线的事故均是由于单个 IDC 因市政施工等原因下线，同时未能成功构建多活 IDC 架构，因此造成 IDC 单点依赖所导致的。

前文也已提到过：由于大部分采用 SOA 架构的产品需要依赖消息中间件来确保系统的最终一致性。因此对其可用性（部分节点故障不会影响正常使用）、可靠性（即使在部分节点故障时，也确保消息不丢失、不重复、并严格有序）、功能性（如：发布/订阅模型、基于轮转的任务分发等）等方面均有较严格的要求。这就必然要用到高可用集群、节点间同步复制、数据持久化等低效率、高维护成本的技术手段。因此消息分发服务也常常成为分布式系统中的一大主要瓶颈。

与 Paxos、Raft 等算法相比，BYPSS 同样提供了故障检测、服务选举、服务发现和分布式锁等分布式协调功能，以及相同等级的强一直性、高可用性和抗脑裂（Split Brain）能力。在消除了几乎全部网络广播和磁盘 IO 等高开销操作的同时，提供了数千、甚至上万倍于前者的访问性能和并发处理能力。可在对网络吞吐和延迟等方面无附加要求的前提下，构建跨多个 IDC 的大规模分布式集群系统。

与各个常见的消息中间件相比，BYPSS 提供了一骑绝尘的单点百万至千万条消息每秒的吞吐和路由能力——同样达到千百倍的性能提升，同时保证消息不重复和严格有序。

然而天下没有免费的午餐，特别是在分布式算法已经非常成熟的今天。在性能上拥有绝对优势的同时，BYPSS 必然也有其妥协及取舍——BYPSS 选择放弃极端（平均每年 2 次，并且大多由维护引起，控制在低谷时段，基于实际生产环境多年统计数据）情形下的可靠性，对分布式系统的具体影响包括以下两方面：

* 对于分布式协调服务来说，这意味着每次发生 BYPSS 主节点故障掉线后，所有的已注册端口都会被强制失效，所有活动的端口都需要重新注册。

  例如：若分布式 Web 服务器集群以用户为最小调度单位，为每位已登陆用户注册一个消息端口。则当 BYPSS 主节点因故障掉线后，每个服务器节点都会得知自己持有的所有端口均已失效，并需要重新注册当前自己持有的所有活动（在线）用户。

  幸运的是，该操作可以被批量化地完成——通过批量端口注册接口，可在一次请求中同时提交多达数百万端口的注册和注销操作，从而大大提升了请求处理效率和网络利用率：在 2013 年出厂的至强处理器上（Haswell 2.0GHz），BYPSS 服务可实现每核（每线程）100 万端口/秒的处理速度。同时，得益于我方自主实现的并发散列表（每个 arena 都拥有专属的汇编优化用户态读者/写者高速锁），因此可通过简单地增加处理器核数来实现处理能力的线性扩展。

  具体来说，在 4 核处理器+千兆网卡环境下，BYPSS 可达成约每秒 400 万端口注册的处理能力；而在 48 核处理器+万兆网卡环境下，则可实现约每秒 4000 万端口注册的处理能力（测试时每个端口名称的长度均为 16 字节），网卡吞吐量和载荷比都接近饱和。再加上其发生概率极低，并且恢复时只需要随着对象的加载来逐步完成重新注册，因此对系统整体性能几乎不会产生什么波动。

  为了说明这个问题，考虑 10 亿用户同时在线的极端情形，即使应用程序为每个用户分别注册一个专用端口（例如：用来确定用户属主、完成消息分发等），那么在故障恢复后的第一秒内，也不可能出现“全球 10 亿用户心有灵犀地同时按下刷新按钮”的情况。相反，基于 Web 等网络应用的固有特性，这些在线用户通常要经过几分钟、几小时甚至更久才会逐步返回服务器（同时在线用户数=每秒并发请求数 x 用户平均思考时间）。即使按照比较严苛的“1 分钟内全部返回”（平均思考时间 1 分钟）来计算，BYPSS 服务每秒也仅需处理约 1600 万条端口注册请求。也就是说，一台配备了 16 核至强处理器和万兆网卡的入门级 1U PC Server 即可满足上述需求。

  作为对比实例：官方数据显示，淘宝网 2015 年双十一当天的日活用户数（DAU）为 1.8

亿，同时在线用户数峰值为 4500 万。由此可见，目前超大型站点瞬时并发用户数的最高峰值仍远低于前文描述的极端情况。即使再提高数十倍，BYPSS 也足可轻松支持。

* 另一方面，对于消息路由和分发服务来说，这意味着每次发生 BYPSS 主节点故障掉线后，所有暂存在 BYPSS 服务器消息队列中，未及发出的待转发消息都将永久丢失。可喜的是，nano-SOA 架构不需要依赖消息中间件来实现跨服务的事务一致性。因此对消息投递的可靠性并无严格要求。

  意即：nano-SOA 架构中的消息丢失最多导致对应的用户请求完全失败，此时仍可保证数据的全局强一致性，绝不会出现“成功一半”之类的不一致问题。在绝大多数应用场景中，这样的保证已经足够——即使支付宝和四大行的网银应用也会偶尔发生操作失败的问题，这时只要资金等帐户数据未出现错误，那么稍候重试即可。

  此外，BYPSS 服务也通过高度优化的异步 IO，以及消息批量打包等技术有效降低了消息在服务器队列中的等待时间。具体来说，这种消息批量打包机制由消息推送和消息发送机制两方面组成：

  BYPSS 提供了消息批量发送接口，可在一次请求中同时提交数以百万计的消息发送操作，从而大大提升了消息处理效率和网络利用率。另一方面，BYPSS 服务器也实现了消息批量打包推送机制：若某节点发生消息浪涌，针对该节点的消息大量到达并堆积在服务器端消息队列中。则 BYPSS 服务器会自动开启批量消息推送模式——将大量消息打包成一次网络请求，批量推送至目的节点。

  通过上述的批量处理机制，BYPSS 服务可大大提升消息处理和网络利用效率，确保在大部分情况下，其服务器端消息队列基本为空，因此就进一步降低了其主服务器节点掉线时，发生消息丢失的概率。

  然而，虽然消息丢失的概率极低，并且 nano-SOA 架构先天就不怎么需要依赖消息中间件提供的可靠性保证。但仍然可能存在极少数对消息传递要求很高的情况。对于此类情况，可选择使用下列解决方案：

  - 自行实现回执和超时重传机制：消息发送方对指定端口发送消息，并等待接收该消息处理回执。若在指定时段内未收到回执，则重新发送请求。

  - 直接向消息端口的属主节点发起 RPC 请求：消息发送方通过端口查询命令获取该端口属主节点的 IP 地址等信息，并直接与该属主节点建立连接、提交请求并等待其返回处理结果。BYPSS 在此过程中仅担当服务选举和发现的角色，并不直接路由消息。对于视频推流和转码、深度学习等有大量数据流交换的节点间通信，也建议使用此方式，以免 BYPSS 成为 IO 瓶颈。

  - 使用第三方的可靠消息中间件产品：若需要保证可靠性的消息投递请求较多，规则也较复杂，也可单独搭建第三方的可靠消息分发集群来处理这部分请求。

    当然，实际上不存在完全可靠的消息队列服务（能保证消息不丢失、不重复、不乱

序）。因此在确实需要实现跨应用服务器节点的分布式事务时，建议通过 BYPSS 以及 BYDMQ 配合 SAGA 等模式来实现，详见：5.2.4 白杨分布式消息队列（BYDMQ）。

综上所述，可以认为BYPSS服务就是为nano-SOA架构量身定做的集群协调和消息分发服务。BYPSS 和 nano-SOA 架构之间形成了扬长避短的互补关系：BYPSS 以极端条件下系统整体性能的轻微波动为代价，极大提升了系统的总体性能表现。适合用来实现高效率、高可用、高可靠、强一致的 nano-SOA 架构分布式系统。

5.2.3.2 BYPSS 特性总结

BYPSS 和基于 Paxos、Raft 等传统分布式一致性算法的分布式协调产品特性对比如下：

项目	BYPSS	ZooKeeper、Consul、etcd…
可用性	高可用，支持多活 IDC。	高可用，但难以支持多活 IDC。
一致性	强一致，主节点通过多数派选举。 读写操作均提供强一致保证。	写入强一致，多副本复制；大部分实现为提升性能，读取时牺牲了一致性（仅 Consul 支持配置成强一致读取模式）。
并发性	千万量级并发连接，可支持数十万并发节点	不超过 5000 节点
容量	每 10GB 内存可支持至少 1 亿消息端口；每 1TB 内存可支持至少 100 亿消息端口；两级并发散列表结构确保容量可线性扩展至 PB 级。	通常最高支持数十万 KV 对。开启了变更通知时则更少。
延迟	相同 IDC 内每次请求延迟在亚毫秒级（阿里云中实测为 0.5ms）；相同区域内的不同 IDC 间每次请求延迟在毫秒级（阿里云环境实测 2ms）。	由于每次请求需要至少三次网络广播和多次磁盘 IO，因此相同 IDC 中的每操作延迟在十几毫秒左右；不同 IDC 间的延迟则更长（详见下文）。
性能	每 1Gbps 网络带宽可支持约 400 万次/秒的端口注册和注销操作。在 2013 年出厂的入门级至强处理器上，每核心可支持约 100 万次/秒的上述端口操作。性能可通过增加带宽和处理器核心数量线性扩展。在现代处理器和双口 4 万兆网卡上可达 3 亿次操作/秒的处理能力。	算法本身的特性决定了无法支持批量操作，相同测试条件下不到 200 次每秒的请求性能（由于每个原子操作都需要至少三次网络广播和多次磁盘 IO，因此支持批量操作毫无意义，详见下文）。
网络利用率	高：服务器端和客户端均具备端口注册、端口注销、消息发送、端口查询、节点查询等原语的批量打包能力，网络载荷比可接近 100%。	低：每请求一个独立包（TCP Segment、IP Packet、Network Frame），网络载荷比通常低于 5%。
可伸缩性	有：可通过级联的方式进行横向扩展。	无：集群中的节点越多（因为广播和磁盘 IO 的范围更大）性能反而越差。
分区容忍	无多数派分区时系统下线，但不会产生广播风暴。	无多数派分区时系统下线，有可能产生广播风暴引发进一步网络故障。
消息分发	有，高性能，客户端和服务器均包含了消息的批量自动打包支持。	无。
配置管理	无，BYPSS 认为配置类数据应交由 Redis、MySQL、MongoDB 等专门的产品来维护和管	有，可当作简单的 CMDB 来使用，这种功能和职责上的混淆不清进一步劣

项目	BYPSS	ZooKeeper、Consul、etcd…
	理。当然，这些 CMDB 的主从选举等分布式协调工作仍可由 BYPSS 来完成。	化了产品的容量和性能。
故障恢复	需要重新生成状态机，但可以数千万至数亿端口/秒的性能完成。实际使用中几无波动。	不需要重新生成状态机。

上述比较中，延迟和性能两项主要针对写操作。这是因为在常见的分布式协调任务中，几乎全部有意义的操作都是写操作。例如：

操作	对服务协调来说	对分布式锁来说
端口注册	成功：服务选举成功，成为该服务的属主。 失败：成功查询到该服务的当前属主。	成功：上锁成功。 失败：上锁失败，同时返回锁的当前属主。
端口注销	放弃服务所有权。	释放锁。
注销通知	服务已下线，可更新本地查询缓存或参与服务竞选。	锁已释放，可重新开始尝试上锁。

上表中，BYPSS 的端口注册对应 ZooKeeper 等传统分布式产品中的“写/创建 KV 对”；端口注销对应“删除 KV 对”；注销通知则对应“变更通知”服务。

由此可见，为了发挥最高效率，在生产环境中通常不会使用单纯的查询等只读操作。而是将查询操作隐含在端口注册等写请求中，请求成功则当前节点自身成为属主；注册失败自然会返回请求服务的当前属主，因此变相完成了属主查询（服务发现/名称解析）等读操作。

需要注意的是，就算是端口注册等写操作失败，其实还是会伴随一个成功的写操作。因为仍然要将发起请求的当前节点加入到指定条目的变更通知列表中，以便在端口注销等变更事件发生时，向各个感兴趣的节点推送通知消息。因此写操作的性能差异极大地影响了现实产品的实际表现。

注：以上所述 nano-SOA 架构和 BYPSS 分布式协调算法均受到多项国家和国际发明专利保护。

5.2.3.3 基于 BYPSS 的高性能集群

从高性能集群（HPC）的视角来看，BYPSS 与前文所述的传统分布式协调产品之间，最大的区别主要体现在以下两个方面：

1. 高性能：BYPSS 通过消除网络广播、磁盘 IO 等开销，以及增加批处理支持等多种优化手段使分布式协调服务的整体性能提升了上万倍。

2. 大容量：每 10GB 内存至少 1 亿个消息端口的容量密度，由于合理使用了并发散列表等数据结构，使得容量和处理性能可随内存容量、处理器核心数量以及网卡速率等硬件升级而线性扩展。

由于传统分布式协调服务的性能和容量等限制，在经典的分布式集群中，多以服务或节点作为单位来进行分布式协调和调度，同时尽量要求集群中的节点工作在无状态模式。服务节点无状态的设计虽然对分布式协调服务的要求较低，但同时也带来了集群整体性能低下等问题。

与此相反，BYPSS 可轻松实现每秒千万次请求的处理性能和万亿量级的消息端口容量。这就给分布式集群的精细化协作构建了良好的基础。与传统的无状态集群相比，基于 BYPSS 的精细化协作集群能够带来巨大的整体性能提升。

我们首先以最常见的用户和会话管理功能来说明：在无状态的集群中，在线用户并无自己的属主服务器，用户的每次请求均被反向代理服务随机地路由至集群中的任意节点。虽然 LVS、Nginx、HAProxy、TS 等主流反向代理服务器均支持基于 Cookie 或 IP 等机制的节点粘滞选项，但由于集群中的节点都是无状态的，因此该机制仅仅是增加了相同客户端请求会被路由到某个确定后台服务器节点的概率而已，仍无法提供所有权保证，也就无法实现进一步的相关优化措施。

而得益于 BYPSS 突出的性能和容量保证，基于 BYPSS 的集群可以用户为单位来进行协调和调度（即：为每个**活动用户**注册一个端口），以提供更优的整体性能。具体的实现方式为：

1. 与传统模式一样，在用户请求到达反向代理服务时，由反向代理通过 HTTP Cookie、IP 地址或自定义协议中的相关字段等方式来判定当前请求应该被转发至哪一台后端服务器节点。若请求中尚无粘滞标记，则选择当前负载最轻的一个后端节点来处理。

2. 服务器节点在收到用户请求后，在本地内存表中检查该用户的属主是否为当前节点。

   a) 若当前节点已是该用户属主，则由此节点继续处理用户请求。

   b) 若当前节点不是该用户的属主，则向 BYPSS 发起 RegPort 请求，尝试成为该用户的属主。此请求应使用批量方式发起，以进一步提高网络利用率和处理效率。

      i. 若 RegPort 请求成功，说明当前节点已成功获取该用户的所有权，此时可将用户信息由后端数据库加载到当前节点的本地缓存中（应使用批量加载优化），并继续处理此用户相关请求。

      ii. 若 RegPort 请求失败，说明指定用户正归于另一个节点管辖，此时应重新设置反向代理能够识别的 Cookie 等粘滞字段，将其指向正确的属主节点。并要求反向代理服务或客户端重试请求。

与传统架构相比，考虑到无状态服务也需要通过 MySQL、Memcached 或 Redis 等有状态技术来实现专门的用户和会话管理机制，因此以上实现并未增加多少复杂度，但是其带来的性能提升却非常巨大，对比如下：

项目	BYPSS HPC 集群	传统无状态集群
1 运维	省去用户和会话管理集群的部署和维护成本。	需要单独实施和维护用户管理集群，并为用户和会话管理服务提供专门的高可用保障，增加故障点、增加系统整体

<table>
<tr><th>项目</th><th>BYPSS HPC 集群</th><th>传统无状态集群</th></tr>
<tr><td></td><td></td><td>复杂性、增加运维成本。</td></tr>
<tr><td>2
网络</td><td>几乎所有客户请求的用户匹配和会话验证工作都得以在其属主节点的内存中直接完成。内存访问为纳秒级操作，对比毫秒级的网络查询延迟，性能提升十万倍以上。同时有效降低了服务器集群的内部网络负载。</td><td>每次需要验证用户身份和会话有效性时，均需要通过网络发送查询请求到用户和会话管理服务，并等待其返回结果，网络负载高、延迟大。

由于在一个典型的网络应用中，大部分用户请求都需要在完成用户识别和会话验证后才能继续处理，因此这对整体性能的影响很大。</td></tr>
<tr><td>3
缓存</td><td>因为拥有了稳定的属主服务器，而用户在某个时间段内总是倾向于重复访问相同或相似的数据（如自身属性，自己刚刚发布或查看的商品信息等）。因此服务器本地缓存的数据局部性强、命中率高。

相较于分布式缓存而言，本地缓存的优势非常明显：
1. 省去了查询请求所需的网络延迟，降低了网络负载（详见本表“项目 2”中的描述）。
2. 直接从内存中读取已展开的数据结构，省去了大量的数据序列化和反序列化工作。
3. 仅由属主节点缓存对应数据也避免了分布式缓存与 DB 之间的不一致问题，提供了强一致保证。

与此同时，如能尽量按照某些规律来分配用户属主，还可进一步地提升服务器本地缓存的命中率。例如：

a) 按租户（公司、部门、站点）来分组用户；
b) 按区域（地理位置、游戏中的地图区域）来分组用户；
c) 按兴趣特征（游戏战队、商品偏好）来分组用户。

等等，然后尽量将属于相同分组的用户优先分配给同一个（或同一组）服务器节点。显而易见，选择合适的用户分组策略可极大提升服务器节点的本地缓存命中率。

这使得绝大部分与用户或人群相关的数据均可在本地缓存命中，不但提升了集群整体性能，还消除了集群对分布式缓存的依赖，同时大大降低了后端</td><td>无专属服务器，用户请求随机到达集群中的任意服务器节点；本地缓存命中率低；各节点重复缓存的内容多；需要以更高的成本为代价依赖分布式缓存。

后端数据库服务器的读压力高，要对其进行分库分表、读写分离等额外优化。

分布式缓存与 DB 之间存在无法避免的数据不一致问题（除非在分布式缓存与 DB 间使用 Paxos 等协议来保证一致性，但随之而来的高昂性能损失也将使分布式缓存失去意义--这比不用分布式缓存还慢）。</td></tr>
</table>

项目	BYPSS HPC 集群	传统无状态集群
	数据库的读负载。	
4 更新	由于所有权确定，能在集群全局确保任意用户在给定时间段内，均由特定的属主节点来提供服务。再加上现代服务器突发宕机故障的概率也较低。 因此可以将用户属性中频繁变化但重要性或时效性较低的部分缓存在内存中，待积累一段时间后再批量更新至数据库。这可大大降低后端数据库服务器的写压力。 例如：商城系统可能随着用户的浏览（比如每次查看商品）进程，随时收集并记录用户的偏好信息。若每次用户查看了新商品后，都需要即时更新数据库，则负载较高。再考虑到因为服务器偶发硬件故障导致丢失最后数小时商品浏览偏好数据完全可以接受，因此可由属主节点将这些数据临时保存在本地缓存中，每积累数小时再批量更新一次数据库。 再比如：MMORPG 游戏中，用户的当前位置、状态、经验值等数据随时都在变化。属主服务器同样可以将这些数据变化积累在本地缓存中，并以适当的间隔（比如：每 5 分钟一次）批量更新到数据库中。 这不但极大地降低了后端数据库要执行的请求数量，而且将多个用户的数据在一个批量事务中打包更新也大大减少数据库操作时的磁盘刷新动作，进一步提升了效率。 此外，由专门的属主节点发起对用户属性的更新也避免了无状态集群中多个节点同时请求更新同一对象时的争抢问题，进一步提高了数据库性能。	由于用户的每次请求都可能被转发到不同服务器节点来处理，因此无法实现累积写入优化和批量写入优化。后端数据库的写负担非常重。 存在多个节点争抢更新同一条记录的问题，进一步加重了数据库负担。 为此要对后端数据库进行额外的分库分表等优化，还会引发“需要由应用层来自行处理分布式事务”等副作用。
5 推送	由于同一用户发起的所有会话均被集中在同一个属主节点内统一管理，因此可非常方便地向用户推送即时通知消息（Comet）。 若发送消息的对象与消息接消息的收用户处于相同节点，则可直接将该消息推送给收件人麾下的所有活动会话。 否则只需将消息定向投递到收件人的属主节点即可。消息投递可使用 BYPSS 实现（直接向收件人对应端口发消息，应启用消息批量发送机制来优化），	由于同一用户的不同会话被随机分配到不同节点处理，因此需要开发、部署和维护专门的消息推送集群，同时专门确保该集群的高性能和高可用性。 这不但增加了开发和运维成本，而且由于需要将每条消息先投递到消息推送服务后，再由该服务转发给客户端，因此也加重了服务器集群的内部网络负载，同时也加大了用户请求的处理延迟。

项目	BYPSS HPC 集群	传统无状态集群
	亦可通过 BYDMQ 等专用的高性能消息中间件来完成。 若按照本表“项目 3”中描述的方法，优先将关联更紧密的用户分配到相同属主节点的话，则可大大提升消息推送在相同节点内完成的概率，此举可显著降低服务器间通信的压力。 因此我们鼓励针对业务的实际情况来妥善定制用户分组策略，合理的分组策略可实现让绝大部分消息都在当前服务器节点内本地推送的理想效果。 例如：对游戏类应用，可按地图对象分组，将处于相同地图副本内的玩家交由同一属主节点进行管理——传统 MMORPG 中的绝大部分消息推送都发生在同一地图副本内的玩家之间（AOI 范围）。 再比如：对于 CRM、HCM、ERP 等 SaaS 应用来说，可按照公司来分组，将隶属于相同企业的用户集中到同一属主节点上——很显然，此类企业应用中，近 100%的通信都来自于企业内部成员之间。 这样即可实现近乎 100%的本地消息推送率，达到几乎消除了服务器间消息投递的效果，极大地降低了服务器集群的内部网络负载。	
6 平衡	集群可使用主被动负载平衡相结合的手段进行调度。 被动平衡：集群中的每个节点均会定期将其麾下不再活跃的用户和会话卸载掉，同时批量通知 BYPSS 服务释放这些用户所对应的端口。此算法实现了宏观上的负载平衡（以较长的时间周期来说，集群是平衡的）。 主动平衡：集群会通过 BYPSS 服务推选出负载平衡协调节点，该节点连续监视集群中各个节点的负载情况，并主动发出指令进行负载调度（如：要求 A 节点将其麾下 5000 位用户的所有权转移给 B 节点）。不同于宏观层面的被动平衡，主动平衡机制可以在更短的时间片内，以迅捷的反应速度来达成集群的快速配平。 主动平衡通常在集群中的部分节点刚刚从故障中	若启用了反向代理中的节点粘滞选项，则其负载平衡性与 BYPSS 集群的被动平衡算法相当。 若未启用反向代理中的节点粘滞选项，则在从故障中恢复时，其平衡性低于 BYPSS 主动平衡集群。与此同时，为了保证本地缓存命中率等其它性能指标不被过分劣化，管理员通常不会禁用节点粘滞功能。 另外，SOA 架构的多个服务间，容易产生负载不平衡，出现一些服务超载，另一些轻载的情况，nano-SOA 集群则无此弊端。

项目	BYPSS HPC 集群	传统无状态集群
	恢复（因此处于空载状态）时效果明显，它比被动平衡反应更加迅速。如：在一个多活 IDC 集群中，某个 IDC 的光缆故障刚刚被修复而恢复上线时。	

值得一提的是，这样的精准协作算法并不会造成集群在可用性方面的任何损失。考虑集群中的某个节点因故障下线的情况：此时 BYPSS 服务会检测到节点已下线，并自动释放属于该节点的所有用户。待其用户向集群发起新请求时，该请求会被路由到当前集群中，负载最轻的节点。这个新节点将代替已下线的故障节点，成为此用户的属主，继续为该用户提供服务（见前文中的步骤 2-b-i）。此过程对用户透明，不需要在客户端中加入额外的处理逻辑。

上述讨论以几乎所有网络应用中都会涉及的用户和会话管理功能为例，为大家展示了 BYPSS HPC 集群精细协调能力的优势。但在多数真实应用中，并不只有用户管理功能。除此之外，应用中通常还会包含可供其用户操作的其它对象。例如在优酷、土豆、youtube 等视频网站中，除了用户以外，至少还有“可供播放的视频”这种对象。

下面我们就以“视频对象”为例，探讨如何使用 BYPSS 的精细化调度能力来大幅提升集群性能。

在这个假想的视频点播类应用中，与前文描述的用户管理功能类似，我们首先通过 BYPSS 服务为每个**活动的视频对象**选取一个属主节点。其次，我们将视频对象的属性分为以下两大类：

1. 普通属性：包含了那些较少更新，并且尺寸较小的属性。如：视频封面和视频流数据在 S3 / OSS 等对象存储服务中的 ID、视频标题、视频简介、视频标签、视频作者 UID、视频发布时间等等。这些属性均符合读多写少的规律，其中大部分字段甚至在视频正式发布后就无法再做修改。

   对于这类尺寸小、变化少的字段，可以将其分布在当前集群中，各个服务器节点的本地缓存内。本地缓存有高性能、低延迟、无需序列化等优点，加上缓存对象较小的尺寸，再配合用户分组等进一步提升缓存局部性的策略，可以合理的内存开销，有效地提升应用整体性能（详见下文）。

2. 动态属性：包含了所有需要频繁变更，或尺寸较大的属性。如：视频的播放次数、点赞次数、差评次数、平均得分、收藏数、引用次数，以及视频讨论区内容等。

   我们规定这类尺寸较大（讨论区内容）或者变化较快（播放次数等）的字段只能由该视频对象的属主节点来访问。其它非属主节点如需访问这些动态属性，则需要将相应请求提交给对应的属主节点来进行处理。

   意即：通过 BYPSS 的所有权选举机制，我们将那些需要频繁变更（更新数据库和执行缓存失效），以及那些占用内存较多（重复缓存代价高）的属性都交给对应的属主节点来管理和维护。这就形成了一套高效的分布式计算和分布式缓存机制，大大提升了应用整体性能（详见下文）。

此外，我们还规定对视频对象的任何写操作（不管是普通属性还是动态属性）均必须交由其属主来完成，非属主节点只能读取和缓存视频对象的普通属性，不能读取动态属性，也不能执行任何更新操作。

由此，我们可以简单地推断出视频对象访问的大体业务逻辑如下：

1. 在普通属性的读取类用户请求到达服务器节点时，检查本地缓存，若命中则直接返回结果，否则从后端数据库读取视频对象的普通属性部分并将其加入到当前节点的本地缓存中。

2. 在更新类请求或动态属性读取类请求到达服务器节点时，通过本地内存表检查当前节点是否为对应视频对象的属主。

    a) 若当前节点已是该视频的属主，则由当前节点继续处理用户请求：读操作直接从当前节点的本地缓存中返回结果；写操作视情形累积在本地缓存中，或直接提交给后端数据库并更新本地缓存。

    b) 若当前节点不是该视频的属主，但在当前节点的名称解析缓存表中找到了与该视频匹配的条目，则将当前请求转发给对应的属主节点。

    c) 若当前节点不是该视频的属主，同时并未在当前节点的名称解析缓存表中查找到对应的条目，则向 BYPSS 发起 RegPort 请求，尝试成为该视频的属主。此请求应使用批量方式发起，以进一步提高网络利用率和处理效率。

        i. 若 RegPort 请求成功，说明当前节点已成功获取该视频的所有权，此时可将视频信息由后端数据库加载到当前节点的本地缓存中（应使用批量加载优化），并继续处理此视频相关请求。

        ii. 若 RegPort 请求失败，说明指定视频对象正归于另一个节点管辖，此时可将该视频及其对应的属主节点 ID 加入到本地名称解析缓存表中，并将请求转发给对应的属主节点来处理。

        注意：由于 BYPSS 能够在端口注销时（无论是由于属主节点主动放弃所有权，还是该节点故障宕机），向所有对此事件感兴趣的节点推送通知。因此名称解析缓存表不需要类似 DNS 缓存的 TTL 超时淘汰机制，仅需在收到端口注销通知或 LRU 缓存满时删除对应条目即可。这不但能够大大增强查询表中条目的时效性和准确性，同时也有效地减少了 RegPort 请求的发送次数，提高了应用的整体性能。

与经典的无状态 SOA 集群相比，上述设计带来的好处如下：

项目	BYPSS HPC 集群	传统无状态集群
1 运维	基于所有权的分布式缓存架构，省去 Memcached、Redis 等分布式缓存集群的部署和维护成本。	需要单独实施和维护分布式缓存集群，增加系统整体复杂性。

项目	BYPSS HPC 集群	传统无状态集群
2 缓存	普通属性的读操作在本地缓存命中，若使用“优先以用户偏好特征来分组”的用户属主节点分配策略，则可极大增强缓存局部性，增加本地缓存命中率，降低本地缓存在集群中各个节点上的重复率。 正如前文所述，相对于分布式缓存而言，本地缓存有消除网络延迟、降低网络负载、避免数据结构频繁序列化和反序列化等优点。 此外，动态属性使用基于所有权的分布式缓存来实现，避免了传统分布式缓存的频繁失效和数据不一致等问题。同时由于动态属性仅被缓存在属主节点上，因此也显著提升了系统整体的内存利用率。	无专属服务器，用户请求随机到达集群中的任意服务器节点；本地缓存命中率低；各节点重复缓存的内容多；需要以更高的成本为代价依赖额外的分布式缓存服务。 后端数据库服务器的读压力高，要对其实施分库分表、读写分离等额外优化。 此外，即使为 Memcached、Redis 等产品加入了基于 CAS 原子操作的 Revision 字段等改进，这些独立的分布式缓存集群仍无法提供数据强一致保证（意即：缓存中的数据与后端数据库里的记录无法避免地可能发生不一致）。
3 更新	由于所有权确定，能在集群全局确保任意视频对象在给定时间段内，均由特定的属主节点来提供写操作和动态属性的读操作等相关服务，再加上现代服务器突发宕机故障的概率也较低。 因此可以将动态属性中频繁变化但重要性或时效性较低的部分缓存在内存中，待积累一段时间后再批量更新至数据库。这可大大降低后端数据库服务器的写压力。 例如：视频的播放次数、点赞次数、差评次数、平均得分、收藏数、引用次数等属性都会随着用户点击等操作密集地变化。若每次发生相关的用户点击事件后，都需要即时更新数据库，则负载较高。而在发生“属主节点由于硬件故障宕机”等极端情况时，丢失几分钟的上述统计数据完全可以接受。因此，我们可以将这些字段的变更积累在属主节点的缓存中，每隔数分钟再将其统一地批量写回后端数据库。 这不但极大地降低了后端数据库收到的请求数量，而且将多个视频的数据在一个批量事务中打包更新，也大大减少数据库操作时的磁盘刷新动作，进一步提升了效率。 此外，由专门的属主节点单独发起对视频记录的更新也避免了无状态集群中多个节点同时请求更新	由于每次请求都可能被路由到不同服务器节点来处理，因此无法实现累积写入优化和批量写入优化。后端数据库服务器的写负担非常重。存在多个节点争抢更新同一条记录的问题，这进一步加重了数据库负担。 为此要对后端数据库进行额外的分库分表等优化，还会引发“需要由应用层来自行处理分布式事务”等副作用。

项目	BYPSS HPC 集群	传统无状态集群
	同一对象时的争抢问题，进一步提高了数据库性能。	
4 平衡	集群可使用主被动负载平衡相结合的手段进行调度。 被动平衡：集群中的每个节点均会定期将其麾下不再活跃的视频对象卸载掉，同时批量通知 BYPSS 服务释放这些视频对应的端口。此算法实现了宏观上的负载平衡（以较长的时间周期来说，集群是平衡的）。 主动平衡：集群会通过 BYPSS 服务推选出负载平衡协调节点，该节点连续监视集群中各个节点的负载情况，并主动发出指令进行负载调度（如：要求 A 节点将其麾下 10000 个视频对象的所有权转移给 B 节点）。不同于宏观层面的被动平衡，主动平衡机制可以在更短的时间片内，以迅捷的反应速度来达成集群的快速配平。 主动平衡通常在集群中的部分节点刚刚从故障中恢复（因此处于空载状态）时效果明显，它比被动平衡反应更加迅速。如：在一个多活 IDC 集群中，某个 IDC 的光缆故障刚刚被修复而恢复上线时。	在从故障中恢复时，其平衡性低于 BYPSS 主动平衡集群。正常情况下则相差不大。 另外，SOA 架构的多个服务间，容易产生负载不平衡，出现一些服务超载，另一些轻载的情况，nano-SOA 集群则无此弊端。

与前文提及的用户管理案例类似，上述精准协作算法不会为集群的服务可用性方面带来任何损失。考虑集群中的某个节点因故障下线的情况：此时 BYPSS 服务会检测到节点已下线，并自动释放属于该节点的所有视频对象。待用户下次访问这些视频对象时，收到该请求的服务器节点会从 BYPSS 获得此视频对象的所有权并完成对该请求的处理。至此，这个新节点将代替已下线的故障节点成为此视频对象的属主（见前文中的步骤 2-c-i）。此过程对用户透明，不需要在客户端中加入额外的处理逻辑。

以上对“用户管理”和“视频服务”案例的剖析均属抛砖引玉。在实际应用中，BYPSS 通过其高性能、大容量等特征提供的资源精细化协调能力可适用于包括互联网、电信、物联网、大数据批处理、大数据流式计算等广泛领域。

5.2.4 白杨分布式消息队列（BYDMQ）

白杨分布式消息队列服务（BYDMQ，读作“by dark”）是一种强一致、高可用、高性能、高吞度、低延迟、可线性横向扩展的分布式消息队列服务。可支持单点千万量级的并发连接以及单点每秒千万量级的消息转发性能，并支持集群的线性横向扩展。

BYDMQ 自身亦依赖 BYPSS 来完成其服务选举、服务发现、故障检测、分布式锁、消息分发等分布式协调工作。BYPSS 虽然也包含了高性能的消息路由和分发功能，但其主要设计目还是

为了传递任务调度等分布式协调相关的控制类信令。而 BYDMQ 则专注于高吞吐、低延迟的大量业务类消息投递等工作。将业务类消息转移到 BYDMQ 后，可使 BYPSS 的工作压力显著降低。

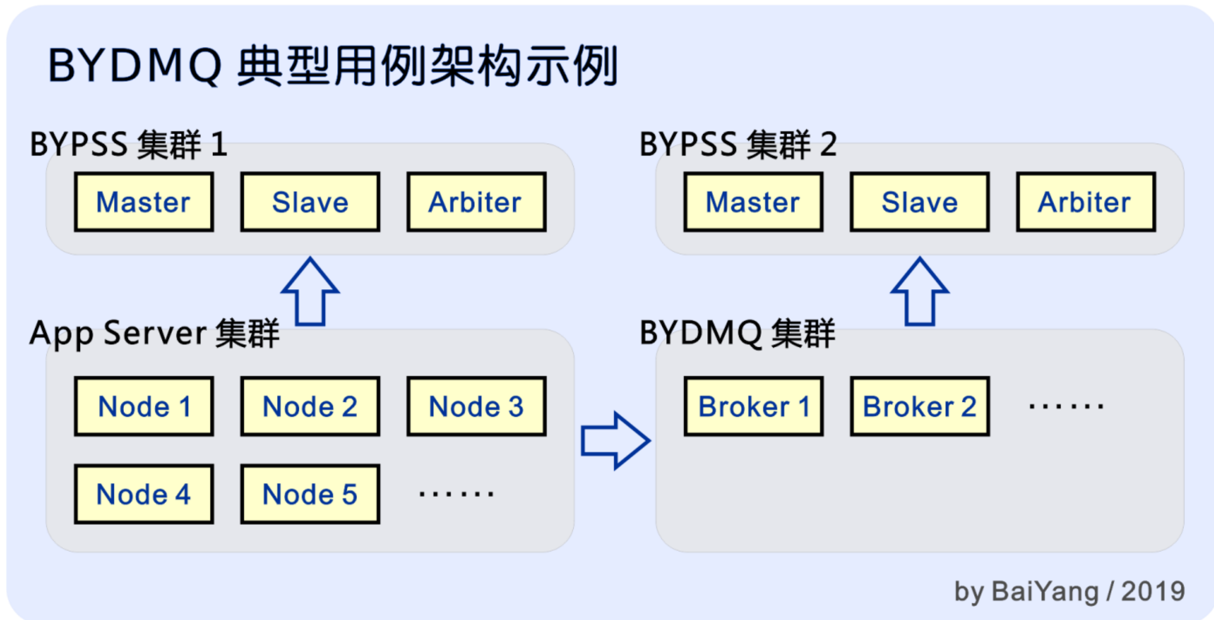


图 27

如图 27 所示，在典型用例中，BYDMQ 与 App Server 集群各自拥有一套独立的 BYPSS 集群，它们分别负责各自的分布式协调任务，App 集群依赖 BYPSS1 完成分布式协调，而其消息通信则依赖 BYDMQ 集群来完成。

不过在研发、测试环境，或业务量不大的生产环境中，也可以让 AppServer 和 BYDMQ 集群共享同一套 BYPSS 服务。另外需要指出的是，此处描述的“独立集群”仅是指逻辑上的独立。而从物理上说，即使是两个逻辑上独立的 BYPSS 集群，也可以共享物理资源。例如：一个 Arbiter 节点完全可以被多个 BYPSS 集群所共享；甚至两个 BYPSS 集群中的 Master、Slave 节点还可以互为主备，这样即简化了运维管理负担，又能够有效节约服务器硬件和能源消耗等资源。

在继续介绍 BYDMQ 的主要特性前，我们首先要澄清一个概念，即：消息队列（消息中间件，MQ）的可靠性问题。众所周知，“可靠的消息传递”包含三个要素——消息在投递过程中，要能够做到不丢失、不乱序和不重复才能称之为可靠。令人遗憾的是，这世间目前并不真正存在同时满足以上三个条件的消息队列产品。或者换句话说，我们目前尚无法在可接受的成本范围内实现同时满足以上三要素的消息队列产品。

要说明这个问题，请考虑以下案例：

图 28

如上图所示，在这一案例中，消息生产者由节点 A、B、C 组成，而消息消费者包含了节点 X、Y、Z，生产者与消费者之间通过一个消息队列连接。现在消息生产者已经生产了 5 条消息，并将它们依序成功提交到了消息队列。

在这样的情形下，我们来逐一讨论消息投递的可靠性问题：

* **消息不丢失**：这点是三要素里最容易保证的。可拆分为两步来讨论：

  - 存储可靠性：可以通过将每条消息同步复制到消息队列服务中的其它节点（Broker）并确保落盘来保证；同时需要使用 Paxos、Raft 等分布式共识协议来确保多个副本间的一致性。但是显而易见，与无副本的纯内存方案相比，由于增加了磁盘 IO、网络复制以及共识投票等步骤，此方案会极大（数千甚至上万倍）地降低消息队列服务的性能。

  - ACK 机制：在生产者向消息队列提交消息，以及消息队列向消费者投递消息时，均增加 ACK 机制来确认消息投递成功。发送方在指定时间内未收到 ACK 机制则重发（重新投递）消息。

  以上两步措施虽可在很大程度保证消息不丢失（至少一次送达），但是可以看到其开销十分巨大，对性能的劣化非常显著。

  与此同时，也应该注意到多副本间的故障检测和主从切换、以及消息收发时的超时重传等技术手段均会引入各自的延迟。而且其中每个步骤中引入的延迟通常都超过数秒钟。

  在真正的用户场景中，这些额外的延迟使得“消息不丢失”的保证除了平白增加了巨大开销以外，大多没有任何实际用处：如今的用户在发起一个请求（如：打开一个链接、提交一个表单等）后，很少有耐心等待许久——若等待数秒后仍然没有响应，他们早就关闭页面离开或 F5 重新刷新了。

  此时无论用户是关闭了页面还是重新发起了请求，已经延迟了几秒（甚至更久）才到达的那条消息（老请求）都已经没有了价值。不但如此，处理这些请求更是在白白消耗网络、运算和存储资源而已——因为其处理结果已经无人问津。

- **消息不重复**：考虑前文提到的 ACK 机制：消费者在处理完一条消息后，需要向消息队列服务回复一条对应的 ACK 信令来确认该消息已被消费。例如：假设上图中的 1 号消息是一个转账请求，消息队列将该消息投递给节点 X 后，节点 X 必须在处理完该转账请求后，向消息队列服务发生一条形如“ACK：Msg=1”的信令来告知消息队列服务，该消息已被处理。而 MQ 无法确保消息不重复的矛盾即在于此：

  仍然按照上例中的假设，MQ 将消息 1 投递到了节点 X，但在规定的时间内却并未收到来自节点 X 的确认（ACK）信令，此时有很多种可能，例如：

  - 消息 1 未被处理：由于网络故障，节点 X 并未收到该消息。
  - 消息 1 未被处理：节点 X 收到了消息，但由于故障掉电而未能及时保存和处理该消息。
  - 消息 1 已被处理：节点 X 收到了并处理了该消息，但由于故障掉电而并能及时向 MQ 服务返回对应的 ACK 信令。
  - 消息 1 已被处理：节点 X 收到了并处理了该消息，但由于网络故障，对应的 ACK 信令未能成功返回至 MQ 服务。

    等等。由此可见，在消息投递超时后，MQ 服务是无法得知该消息是否已被消费的。雪上加霜的是，由于前文所述的原因（不能让用户等太久），这个超时通常还要设置的尽量短，这就让 MQ 服务正确感知实际情况变得更加不可能。

    此时为了保证消息不丢失，MQ 通常会假设消息未被处理，而重新分发该消息（例如在超时后，将消息 1 重新分发给节点 Y）。而这就势必无法再保证消息不重复了，反之亦然。

- **消息不乱序**：从上例中可以看出，所谓“消息不乱序”是指 MQ 中的消息要按照先来后到，以“1、2、3、4、5”的顺序逐一被消费。要保证严格的不乱序，就要求 MQ 必须等待一条消息处理结束（收到 ACK）后，才能继续分发队列中的下一条消息，这至少带来了以下问题：

  - 首先，MQ 中的多条消息无法被并行地消费。例如：MQ 无法将消息 1、2、3 同时分别派发给节点 X、Y、Z，这使得大量消费者节点长期处于饥饿（空闲）状态，甚至于即使在正在执行消息处理的节点上（比如节点 X）也会有大量处理器核心、SMT 单元等计算资源被浪费。
  - 其次，在处理一条消息的过程中，所有其后续消息均只能处于等待状态。若一条消息投递失败（超时），则在其“超时-重新投递”期间内，则会长时间阻塞其所有后续消息，使得它们无法被及时处理。

    由此可见，保证消息严格有序会极大地影响系统整体的消息处理性能、增加硬件采购和运维成本，同时也会显著破坏用户体验。

由以上论述可知，现阶段尚无在合理成本下提供消息可靠传递的 MQ 产品问世。在此前提下，

目前的解决方案主要是依赖 App Server 自身的业务逻辑（如：等幂操作、持久化状态机等）算法来克服这些问题。

反过来说：无论使用号称多“可靠”的 MQ 产品，现在的 App 业务逻辑中也均需要处理和克服上述种种消息投递不可靠的情形。既然 MQ 本质上做不到消息可靠，同时 App 也已经克服了这些不可靠性，那又何必再花费性能被劣化几千、甚至几万倍的代价来在 MQ 层实现支持“分布式存储 +ACK 机制”的方案呢？

基于上述思想，BYDMQ 并不像 RabbitMQ、RocketMQ 等产品那样，提供所谓（实际无法达到）的“可靠性”保证。相反，BYDMQ 采用“尽力送达”的模式，仅在确保不损失性能的前提下，尽可能地保证消息被可靠送达。

正如前文所述，由于 App 已经克服了消息传递过程中偶尔出现的不可靠。因此这样的设计抉择在极大提升了系统性能之余，并未实际增加业务逻辑的开发工作量。

基于上述设计理念的 BYDMQ 包含了以下特性：

* 与 BYPSS 一样基于白杨应用支撑平台中的各优质跨平台组件实现，如：单点支持千万量级并发的网络服务器组件；支持多核线性扩展的并发散列表容器等等。这些优质高性能组件使得 BYDMQ 在可移植性、可扩展性、高容量、高并发处理能力等方便均拥有非常好的表现。

* 与 BYPSS 一样，在客户端和服务器端均拥有成熟的消息批量打包机制。支持连续消息的自动批量打包，大大提高网络利用率和消息吞吐量。

* 与 BYPSS 一样支持 pipelining 机制：使得客户端无需等待一条指令的响应结果即可连续发送下一条指令，显著降低了命令处理延迟、提高了网络吞吐、有效增加了网络利用率。

* 每个客户端（App Server）可注册一个专属 MQ，并通过长连接 + 心跳的方式保活，对应的属主 Broker（BYDMQ 节点）也通过此长连接向客户端实时推送到达的消息（带有批量打包机制）。

* 客户端通过一致性散列算法来推测一个 MQ 的属主（Broker），Broker 在首次收到针对指定 MQ 的请求（如：注册、发消息等）时，将通过 BYPSS 服务来竞选成为该 MQ 的属主。若竞选成功则继续处理，竞选失败则引导客户端重新连接到正确的属主节点。

  与此同时，BYDMQ 会通过 BYPSS 服务实时感知集群变化（如：现有 Broker 下线、新的 Broker 上线等），并将这些变化实时推送到每个客户端节点。这就保证了除非 BYDMQ 集群正在发生剧烈变化（大量 Broker 节点上线或下线等），否则通过一致性散列算法推定属主的准确性是非常高的，从而基本无需再次重定向请求。

  此外，即使一致性散列算法推定错误，该 MQ 的实际属主也会被客户端节点自动记忆到本地快查表中，确保下次向这个 MQ 发送消息时能够直接投递到正确的 Broker。

  这种由客户端直接向对应 MQ 之属主（Broker）投递消息的方法避免了服务器集群中的复杂路由和消息的多次中转，将消息投递的网络路由降至最简，极大地提升了消息投递的效率，有效降低了网络负载。

通过 BYPSS 来为 MQ 选举属主节点的做法则为集群提供了“每个 MQ 在全局范围内唯一”的一致性保证。与此同时，BYPSS 服务也负责在各个 Broker 节点之间分发一些控制类信令（如：节点上下线通知等等），使 BYDMQ 集群可以被更好地统一协调和调度。

* 所有待分发的消息仅存储在对应 MQ 属主（Broker）节点的内存中，避免了写盘、复制、共识投票等大量无用开销。

* 发送方可以为每条消息分别设置其生命期（TTL）和发送失败时的最大重试次数。可根据消息的类型和价值精确控制其消耗的资源。对时效性短或不重要的请求，可及时使其失效，避免在各个环节继续空耗资源，反之亦然。

* 支持分散投递：当指定 MQ 所属客户端节点未上线，或其连接断开超过指定的时长后，此消息队列中的所有待投递消息将被随机发送到任意一个仍可正常工作的 MQ 中。

  分散投递方案预期客户端（App Server）也是一个基于 BYPSS 的 nano-SOA 架构集群。此时若一个 App Server 节点因为运维、硬件故障或网络分区等原因下线，则该节点下辖的所有对象都会被管理该集群的 BYPSS 释放。

  此时系统可随机将发往此节点的请求散布至其它仍正常工作的节点，可让这些目标节点通过 RegPort 重新获取该请求相关对象的所有权，并接替其已经下线的原属主节点继续完成处理。这就大大降低了在一个 App Server 节点异常下线的瞬间，与之相关请求的失败率，优化了客户体验。同时随机散布也使得下线节点麾下的对象被集群中仍然工作的其它节点均分，保证了集群的负载平衡。

综上，BYDMQ 通过在一定程度上牺牲了本就无法真正保证的消息可靠性，再配合消息打包、pipelining、属主直接投递等方式，极大地提升了消息队列服务的单点性能。同时得益于 BYPSS 为其引入的强一致、高可用、高性能的分布式集群计算能力，使其拥有了优异的线性横向扩展能力。加之其对每条消息的灵活控制、以及分散投递等特性，最终为用户提供了一款超高性能的高品质分布式消息队列产品。

5.3 安全隧道服务（BYST）

白杨安全隧道服务（BYST，读作“best”）为用户提供了一种端到端的安全隧道服务，主要支持以下特性：

1. 支持 PSK 和 PKI 鉴权：可使用预分配密钥和公钥体系架构进行鉴权。

2. 支持数十种强加密算法：支持数十种块加密和流式加密算法。

3. 支持实时数据压缩：基于 lz4 算法的高性能实时数据压缩，可通过配置选项开启。

4. 支持消息完整性校验：使用高性能散列算法计算校验和，以确保消息的可靠性和完整性（校验和与压缩后的数据均会被加密后再传输）。此功能可通过配置选项开启。

5. 支持防回放攻击：可通过配置项启用并设置防回放攻击的时间窗口范围。

6. 数据混淆：不同于 OpenVPN、SSL、SSH、IPSec 等现有方案，本隧道协议无任何可被观测的特征。不知道密钥的第三方拦截者只能观察到随机二进制字节流，难以通过任何有效手段侦测出通信双方正在使用本隧道协议。

   不但自己无特征，BYST 还可以扮演 http 等合法白名单协议（BYST over http），即便是在某些仅允许白名单协议的极端环境下，也可正常工作。

   与此同时，BYST 还支持 HTTP chunked transfer encoding，可以做到每个 package（几十 KB~几 MB）仅增加 3.5 个 Bytes 左右的开销，数据膨胀率低于 0.01%，避免了笨重的 HTTP 包头导致数据膨胀的问题。远远低于 SSL/TLS、SSH、h2、WebSocket 等其它白名单协议所带来的额外通信开销，在实现白名单通信的同时仍维持了极高的网络利用率。

7. 高效率：得益于基于汇编优化、零内存拷贝以及 DMA＋ 硬件中断的纯异步 IO 通信组件（详见：1.4.1 单点支撑千万量级并发的高效 IO 服务器组件），即使在受限的硬件平台上，BYST 也可以满足高性能和高并发业务的严苛需求。

   此外，得益于我方成熟的 IO 批量打包、专利的分布式 N:M 动态连接池加速，以及高性能数据实时压缩等算法，BYST 显著提升了网络利用率（高载荷比）和网络吞吐量。

   与此同时，相比于各类现有方案，已将握手（协商）、挥手、确认等环节进行了精心化简的 BYST 隧道协议亦明显具备更低的通信延迟。

8. 高性能和高可用集群：支持基于 BYPSS（详见：1.4.2 强一致多活 IDC 高可用（HAC）和高性能（HPC）服务器集群）的高性能和高可用多活 IDC 集群部署。

   此外还支持多路径自动验活和路由自动切换等功能。可通过增加备用链路来显著提升服务整体可用性，同时也支持多个链路之间的自动流量聚合及负载平衡等功能。

9. 线路级实时补偿：独有的线路级实时跟踪和预测智能算法，对各线路的丢包和抖动分别进行实时跟踪和补偿。

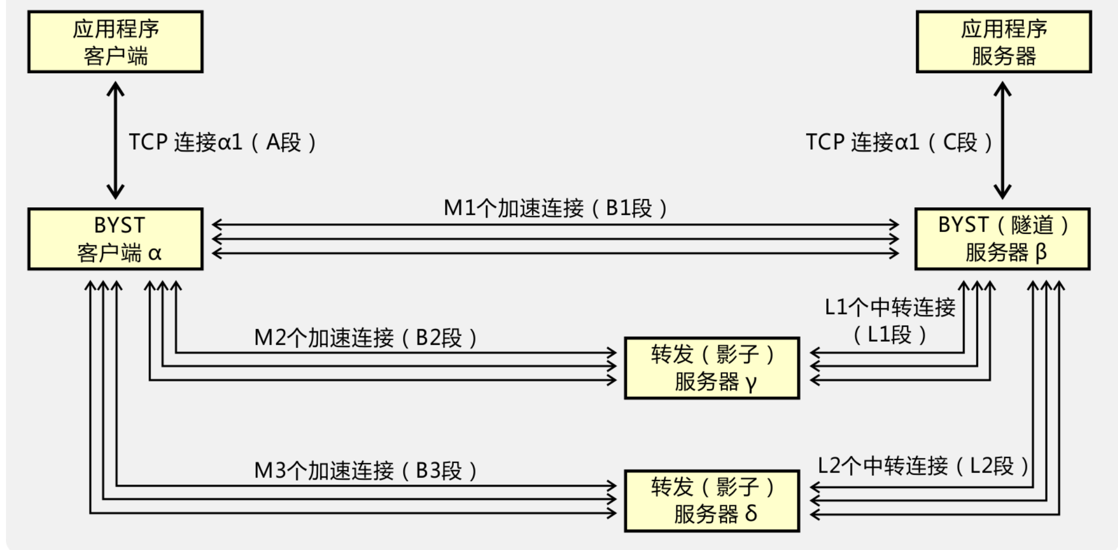
BYST 服务主要用于在 Internet、卫星、微波、SDH（MSTP）、以及跨区域光纤专线等广域或城域网环境中建立安全可靠的数据传输通道，通过强加密和强校验算法来保证数据的安全性、可靠性和完整性，并通过数据压缩降低带宽成本。同时还可通过难以被分析的混淆算法来保护其隧道通信不会被识别、拦截和屏蔽。

综上所述，BYST 主要带来如下技术优势：

1. 保障通信安全：提供带有强加密、强校验以及防回放攻击机制的安全通信隧道。

2. 打满带宽：得益于独有的IO自动批量打包、实时数据压缩，以及**我方专利的分布式 N:M:N 动态连接映射加速算法**，BYST 可做到打满用户带宽上限，显著提升 Site-to-Site Tunnel

通信性能。大量用户实测表明，在同城通信（MAN）环境中，BYST 仅启用单点 N:M 加速即可提升超过 6 倍的带宽吞吐；而在异地通信（WAN）环境中，BYST 的单点 N:M 加速则可实现高达 70 倍的吞吐性能提升。而分布式 N:M 加速则可在此基础上随着分布式加速节点的增加而线性提升聚合吞吐率。



3. 防止误杀：正如前文所述，为提高通信性能以及降低握手延迟，BYST 本身被设计为完全无特征的通信协议。此外，通过数据强加密、IO 自动批量打包、实时数据压缩、以及我方专利的分布式 N:M:N 动态连接映射加速算法等数据混淆机制，BYST 所承载的所有上层协议也将失去其可识别的特征。而 BYST over HTTP 的白名单协议通信支持则进一步保证了其优秀的防火墙通过性。

4. 代理鉴权：BYST 可通过 PSK、PKI 以及 EAL5+ 等级的安全智能卡硬件等技术提供安全可靠的双端鉴权访问，从而免除了上层应用间相互对接时还需要自行实现 CHAP、IKE、LDAP 等复杂算法来完成鉴权的麻烦。

6．总结

事实上，蓝鲸来自客户而不是实验室，所有的特别之处均来自不同客户真实需求，蓝鲸计费力求更加贴近客户，而且可以灵活地满足各种类型的用户。

对于中小企业(SMB)及酒店，蓝鲸计费提供了 IP 架构的支持，对 VoIP 和 TDM 无缝支持，超大话务量处理，以及对内存管理的优化，计费端作为 Server 而减轻 PBX 的负担；最终用户可以根据需求购买必要的计费模块，无需高额费用即可使用高性能的计费系统。

对于联络中心，由于作为利润中心的联络中心，精准计费更便于利润核算，具备分拣和拆帐功能，可以针对每个分机在不同通话类型中进行统计，便于外包型呼叫中心的成本管理。

对于多个分支机构的大型企业，由于蓝鲸计费就是针对分布式 IPT 而设计的企业级计费及成本管理系统，其最大特点是适用于通过 IP 组网的多分支大型企业，独特的"话费节省"统计功能更可以明显得出使用 IPPBX 前后的话费总额对比。

蓝鲸计费作为一套企业级的成本管理系统，以其特有的高性能，以及融入 IPT 系统的能力，完善地支持了当今各种大型而复杂的 IP 语音应用。