软件架构与中间件

涂志莹
tzy_hit@hit.edu.cn

哈尔滨工业大学

软件架构与中间件

Software Architecture and Middleware

第3章
计算层的软件架构技术

第3章 计算层的软件架构技术

3.1 计算层的软件架构技术挑战

3.2 分布式编程模型

3.3 负载均衡

3.4 消息队列

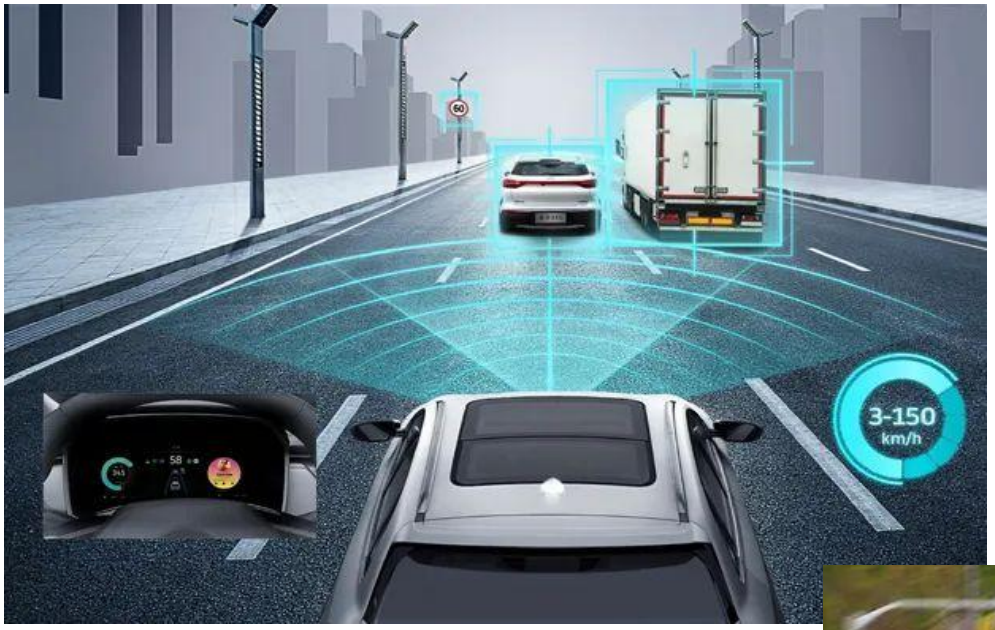
3.5 分布式服务框架

3.1 计算层的软件架构技术挑战

- 计算模式的变革
- 非功能度量指标

计算模式的变革

计算复杂性（例子：无人车）

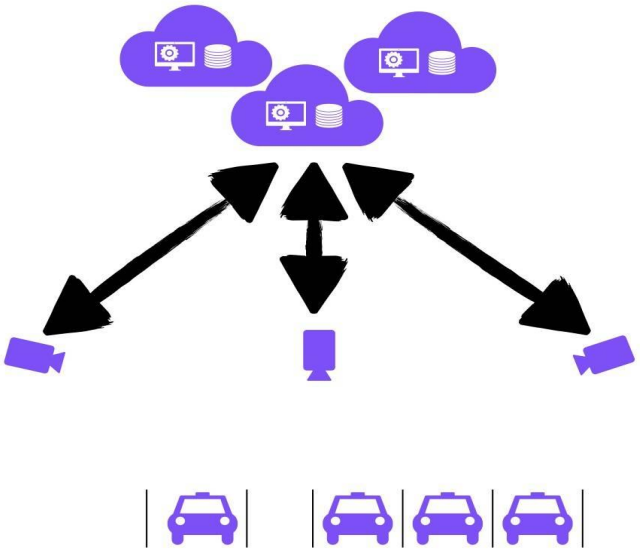


无人车智能识别

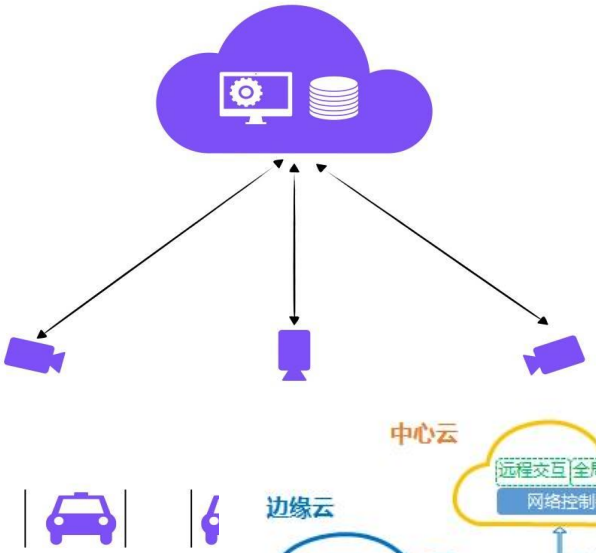
多车协同与车路协同

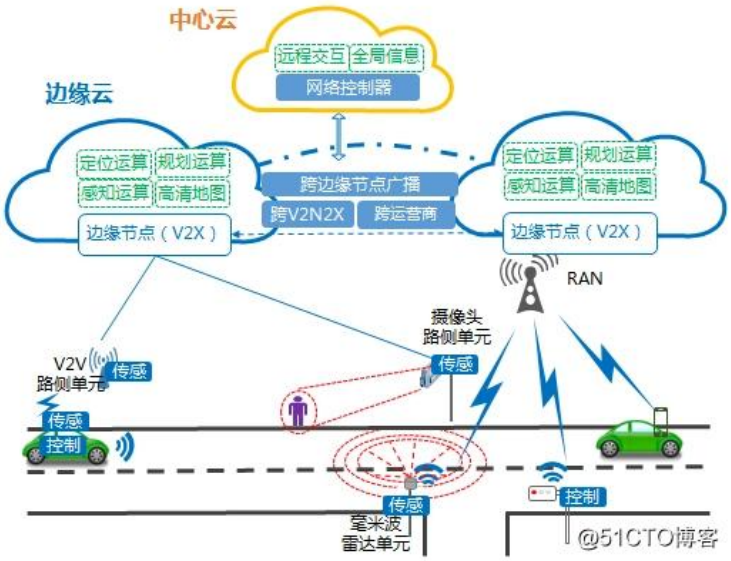
计算复杂性（例子：无人车）

Cloud Computing



Edge Computing





计算复杂性

- 单机任务
  - 手机、车载终端、摄像头、音箱等智能终端
- 想快点
  - 升级处理器
  - 并行计算
- 想脱离客户机/多用户
  - 服务器
  - 云识别服务
- 用户规模激增(e.g. 100W)
  - 集群
  - 消息队列

高性能

高扩展

计算复杂性

- 用户规模激增(e.g. 100W)
  - 集群
  - 消息队列
  - 负载调度
- 用户规模爆炸(e.g. 10000W+)
  - 多机房
  - 冗余
  - 异地多活
  - 降级等

高可用

计算复杂性

Hardware Define Software Era

The limited capacity of hardware resources limits the ability and form of software.

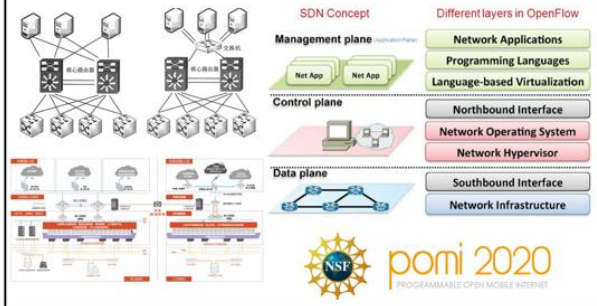
Mainframe

Minicomputer

Web

Software Define Everything Era

Programmable management of hardware based infrastructure resources; Separation of data and control, application programming definition management.

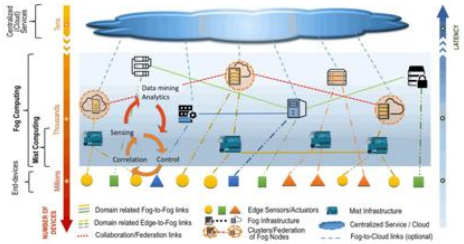


Distributed Service Cluster

Cloud

Service resource collaboration Era

Hardware based infrastructure resources can be coordinated throughout the network; Ubiquitous deployment service resources can be coordinated throughout the network.



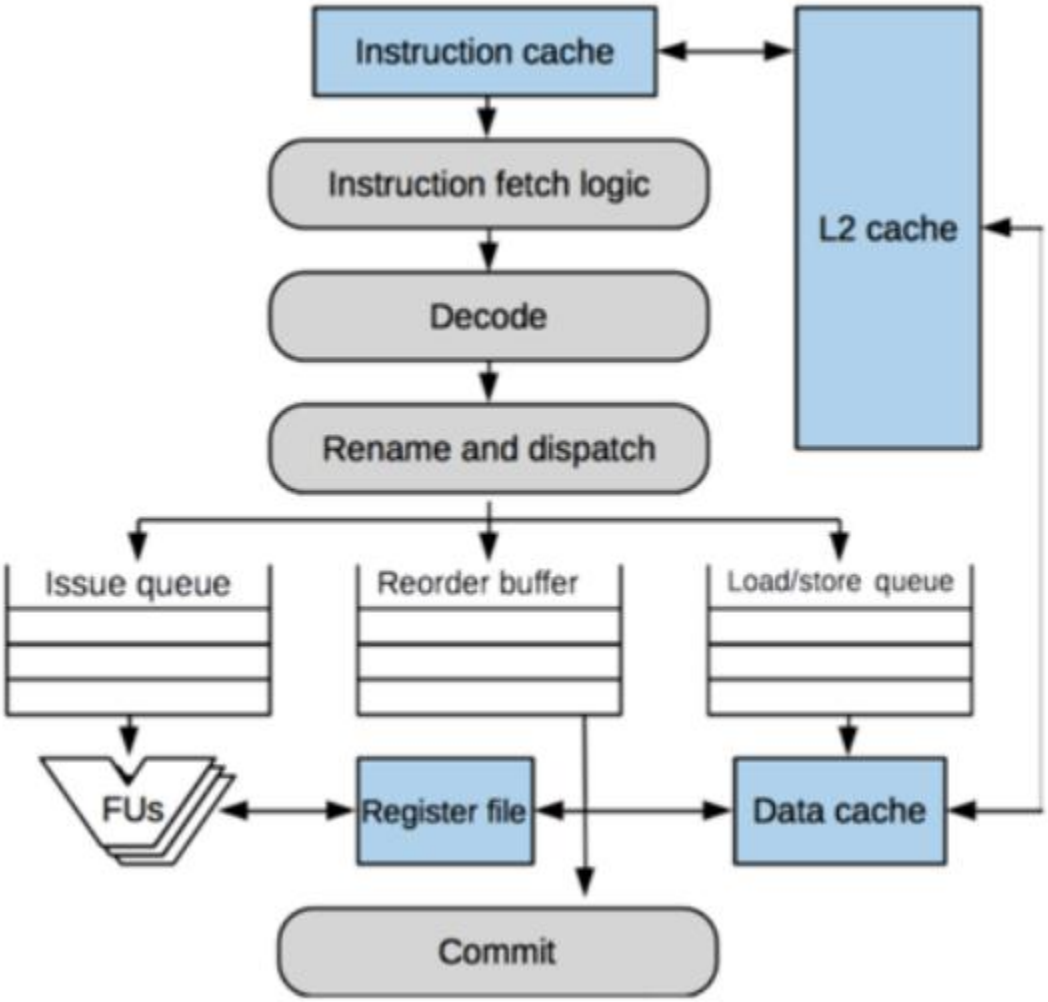
Mobile Network

IoT

Edge

计算模式变革

- 单机计算模式-串行



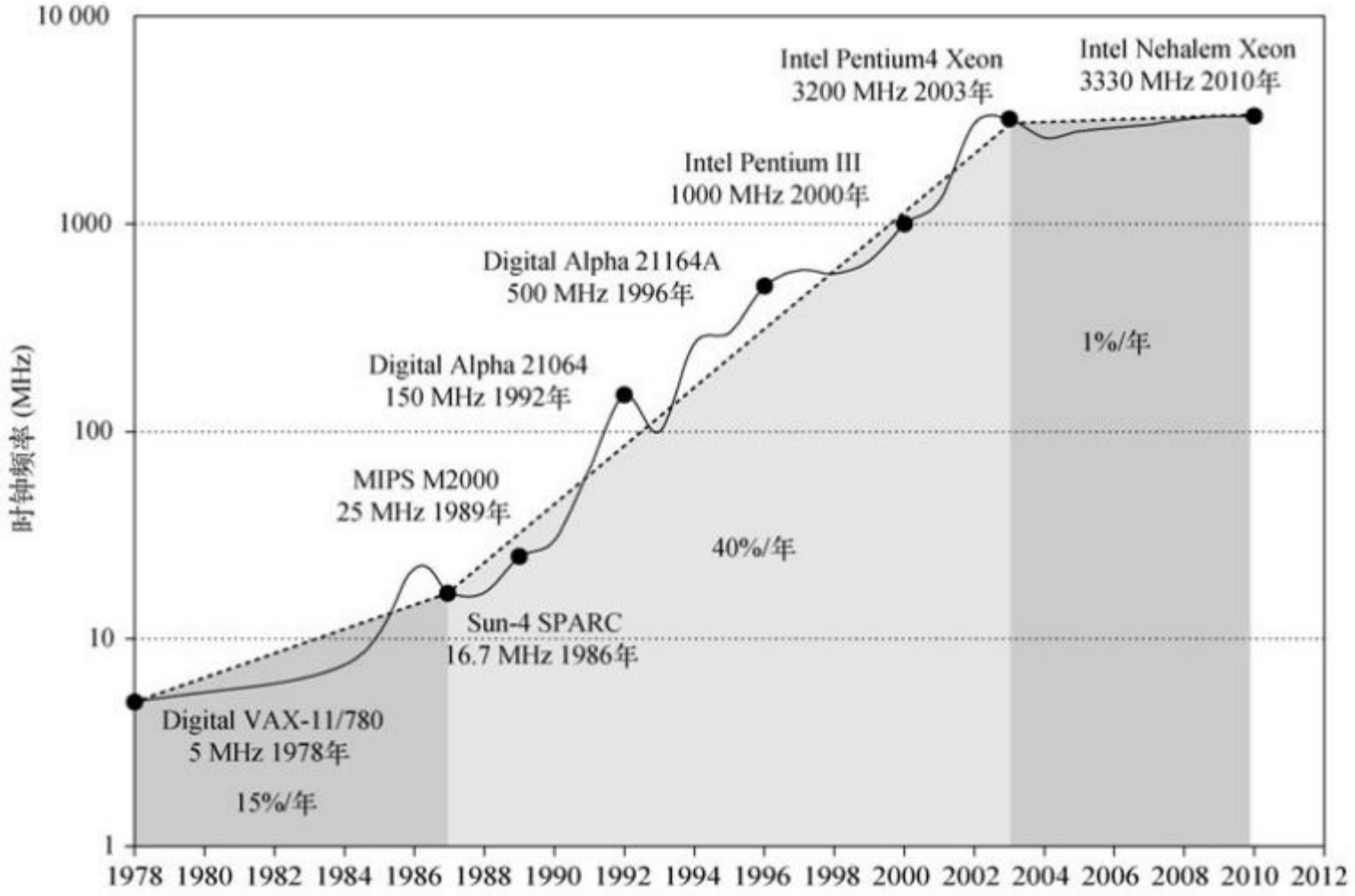
1. R1=M[1024]
2. R1=R1+2
3. M[1032]=R1
4. R1=M[2048]
5. R1=R1+4
6. M[2056]=R1

寄存器重命名

1. R1=M[1024]	4. R2=M[2048]
2. R1=R1+2	5. R2=R2+4
3. M[1032]=R1	6. M[2056]=R2

计算模式变革

- 摩尔定律的高与胖

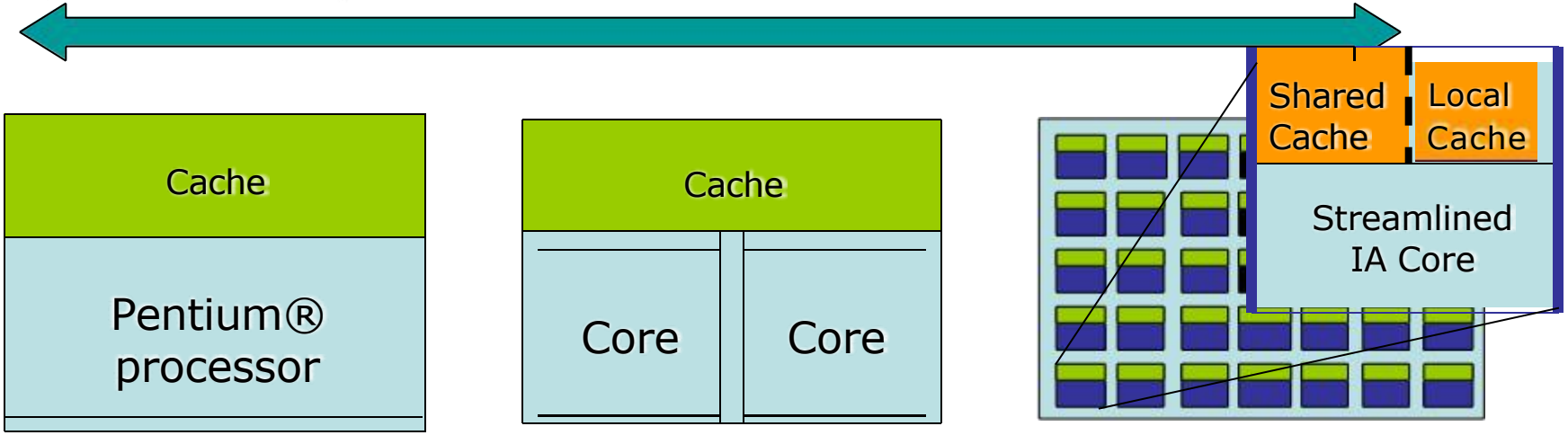


计算模式变革

- 单机计算模式-多核、众核计算

Optimized for speed

Optimized for performance/watt



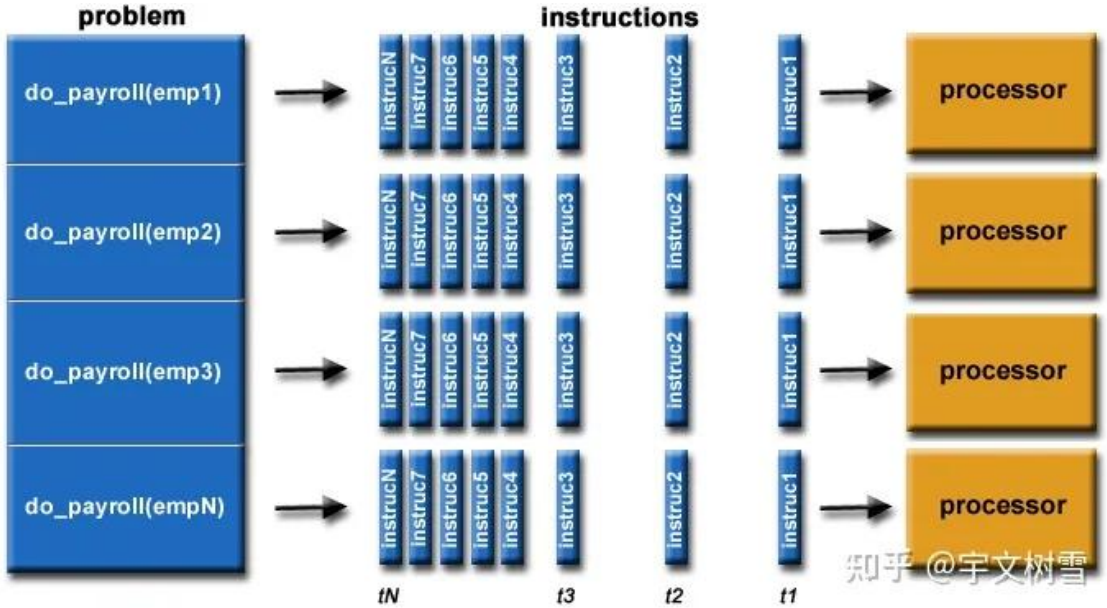
Pentium® processor era chips optimized for raw speed on single threads. Pipelined, out of order execution

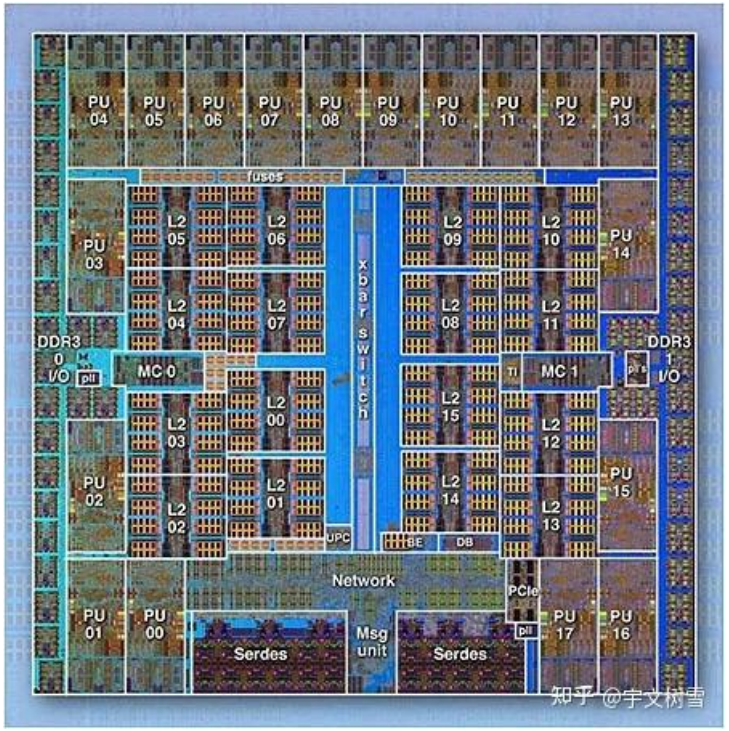
Today's chips use cores which balance single threaded and multi-threaded performance

5-10 years: 10s-100s of energy efficient, IA cores optimized for Multi-threading

计算模式变革

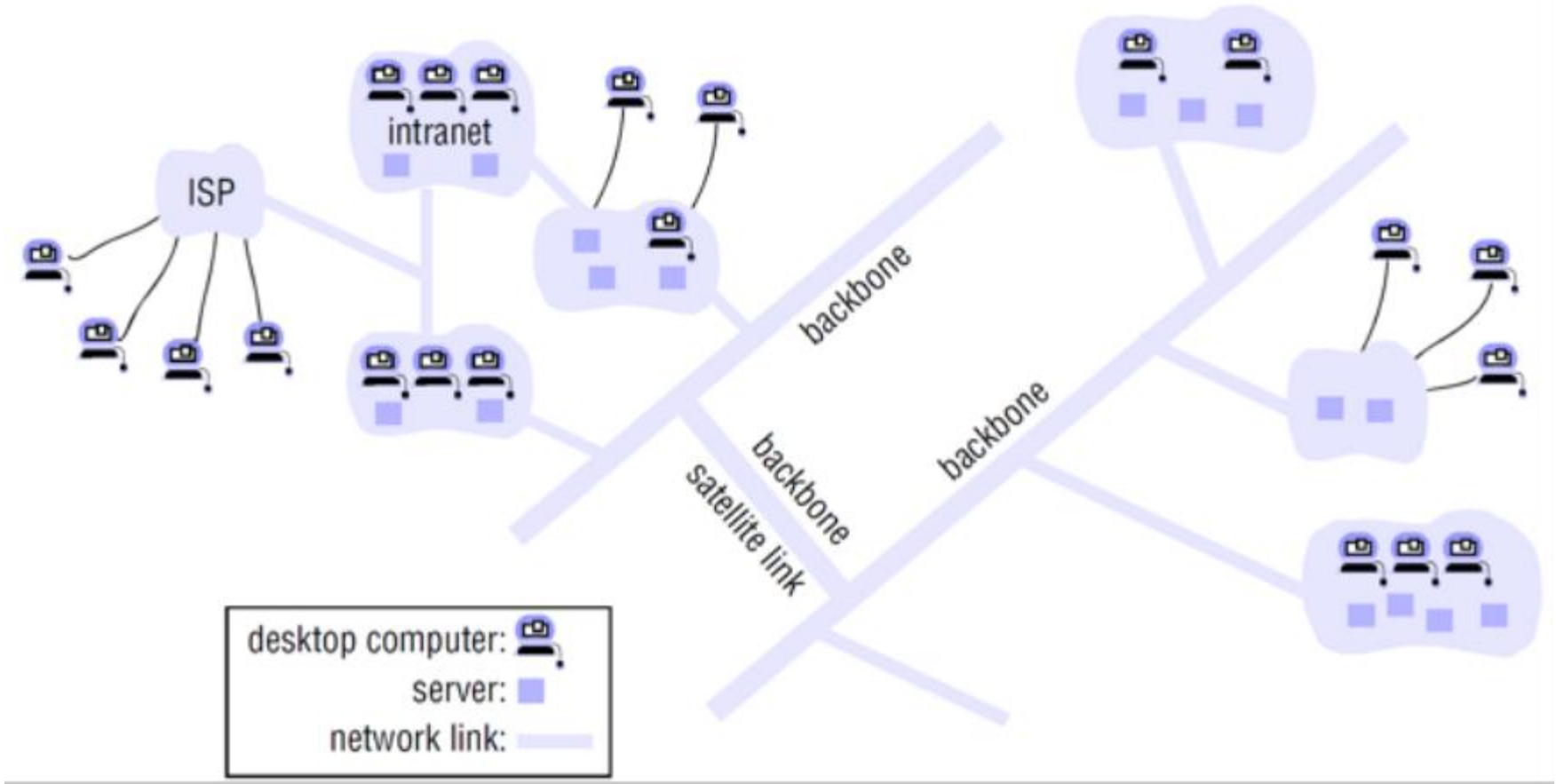
- 单机计算模式-多核、众核计算





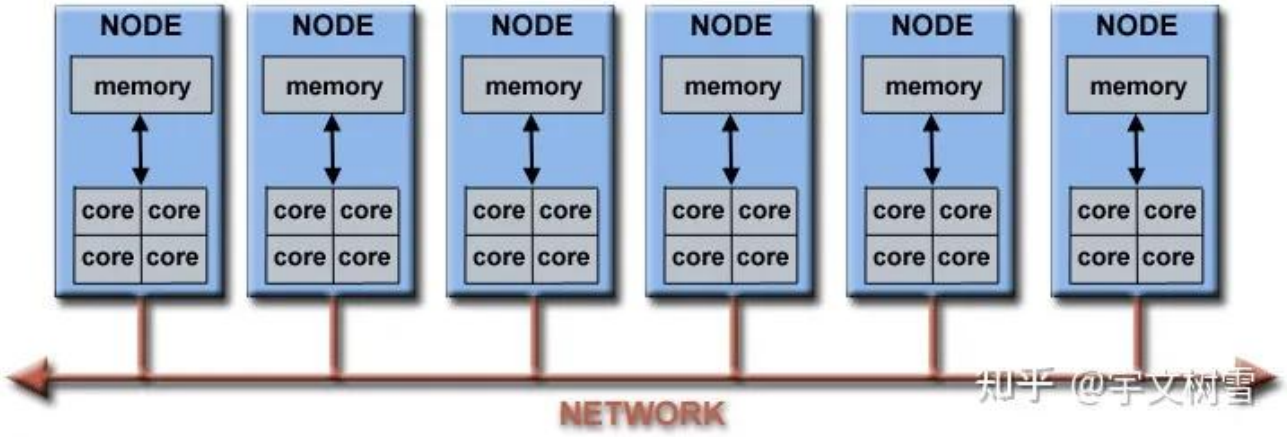
计算模式变革

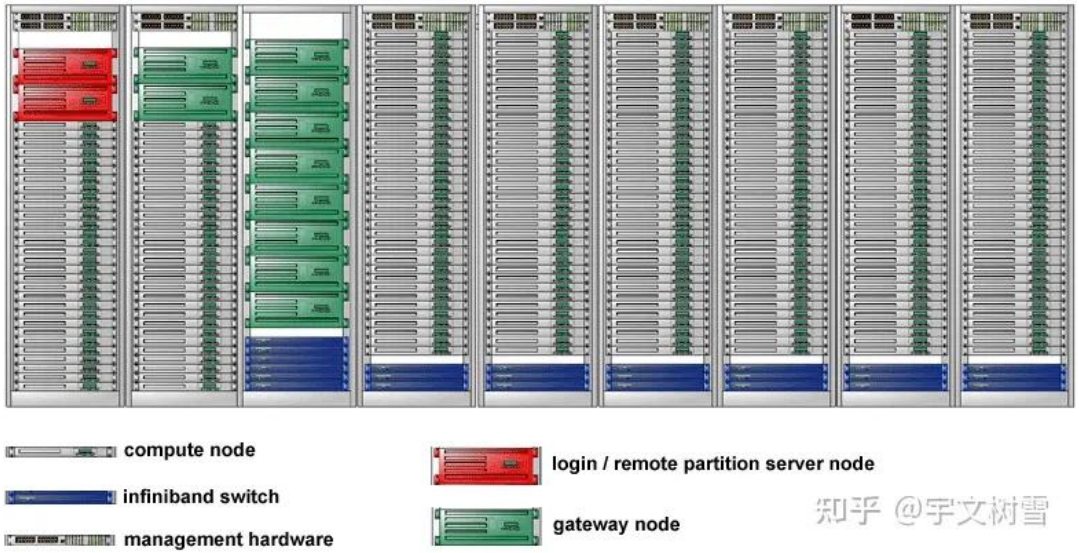
- 分布式计算模式



计算模式变革

- 分布式计算模式





分布式服务的变革

- 大型应用中业务需求的爆炸式增长
- 技术的不断演进导致系统异构化严重
- 业务与技术的沟通存在的鸿沟

传统垂直架构改造的核心就是要对应用进行服务化，
服务化改造使用到的核心技术就是分布式服务框架。

应用从集中走向分布式

- 依旧存在的问题：
  - 系统开发维护成本高，部署效率低，应用数量膨胀，数据库连接数持续变高
  - 代码复用难，导致开发、测试、维护等工作烦、难、杂
  - 难以适应敏捷持续交付的挑战

负载均衡器

负载分发

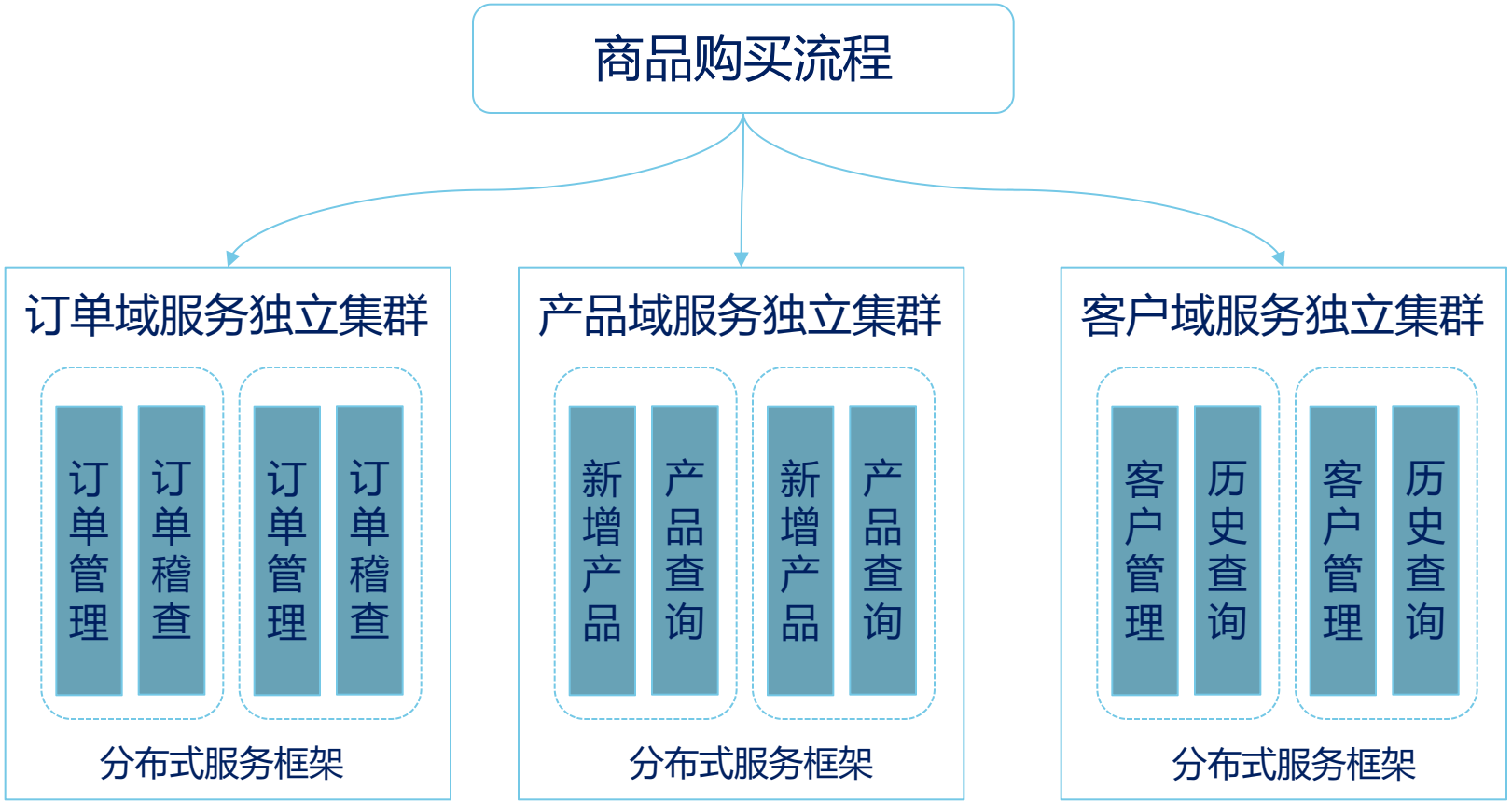
集群服务器1

集群服务器2

大规模系统架构的设计一般原则就是尽可能地拆分，以达到更好的独立扩展与伸缩、更灵活的部署、更好的隔离和容错、更高的开发效率。

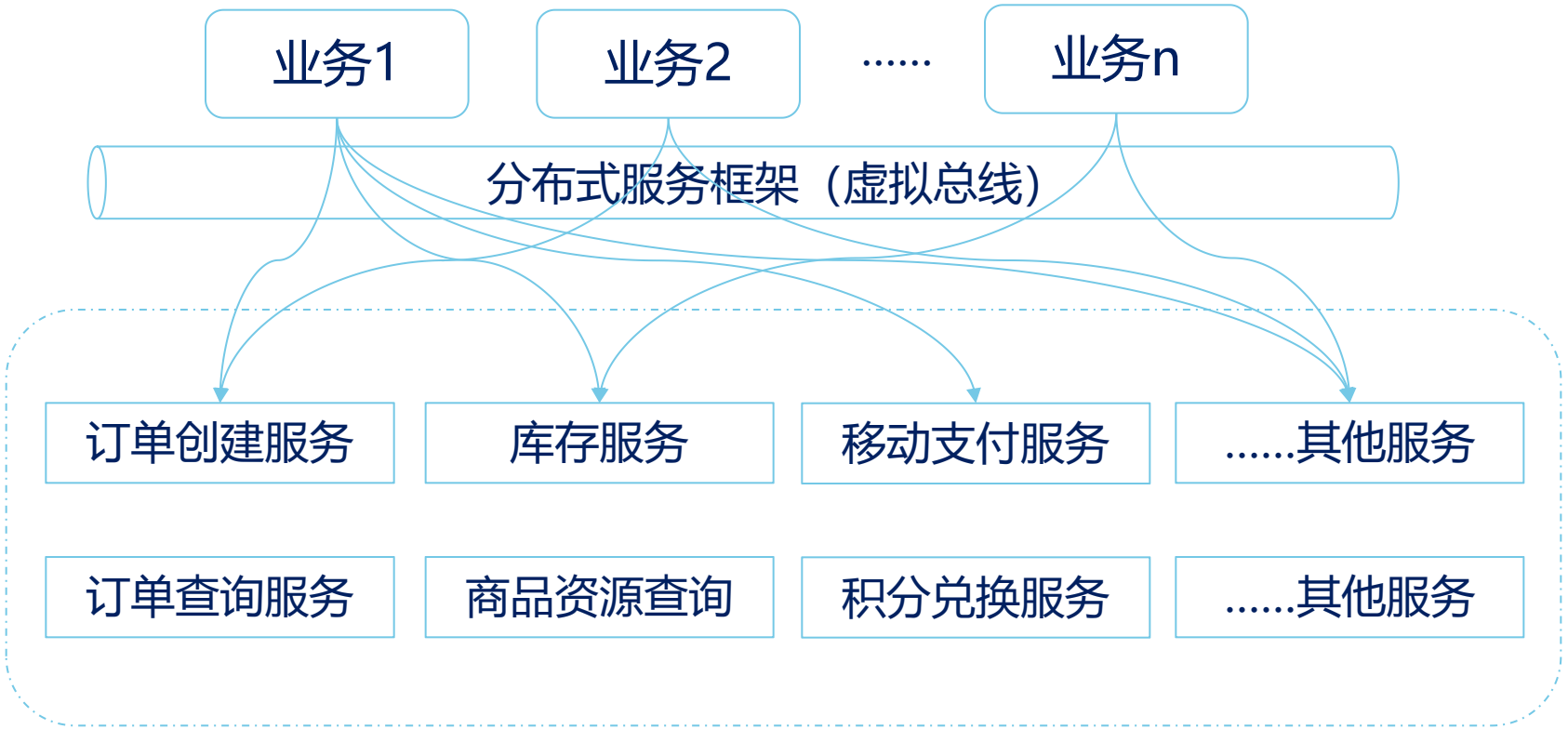
业务纵向拆分

按业务进行梳理，根据业务的特性把应用拆开，不同的业务模块独立部署。



业务横向拆分

将核心的、公共的业务拆分出来，通过分布式服务框架对业务进行服务化，消费者通过标准的契约来消费这些服务。服务提供者独立打包、部署和演进，与消费者解藕。



梦魇依旧——服务治理

- 随着业务拆分，服务数增多，服务治理问题成为了亟待解决的问题。
- 服务治理的目标：有效管控服务，提升服务运行质量，防止业务服务代码架构腐化。
- 服务治理要解决的主要问题如下：
  - 生命周期的管理：服务上线随意，线上服务鱼龙混杂，上线容易下线难。需要规范化上线审批、测试发布流程、下线通知等。
  - 服务容量规划：需要能够采集服务调用性能、时延、成功率、系统资源占用等综合性能指标，分析并识别服务容量瓶颈，合理分配服务容量（计算、数据、网络等资源）
  - 运行时治理：流量陡增时的非核心业务SLA降级；缓存失效，数据I/O开销陡增导致业务失败，需要在线调大服务调用超时时间；非核心服务故障时，业务应放行并执行本地降级逻辑。
  - 服务安全：身份验证与权限验证。

非功能度量指标

性能度量

- CPU速度(MIPS Million Instructions Per Second)
- 网络带宽(Mbps Megabits per second)
- 吞吐量(MIPS、TFLOPS、TPS、QPS)
  - 指系统在单位时间内处理请求的数量。
- RT响应时间(s)
  - 指系统对请求作出响应的时间。
- 网络延时(s)
- 并发用户数
  - 指系统可以同时承载的正常使用系统功能的用户的数量。

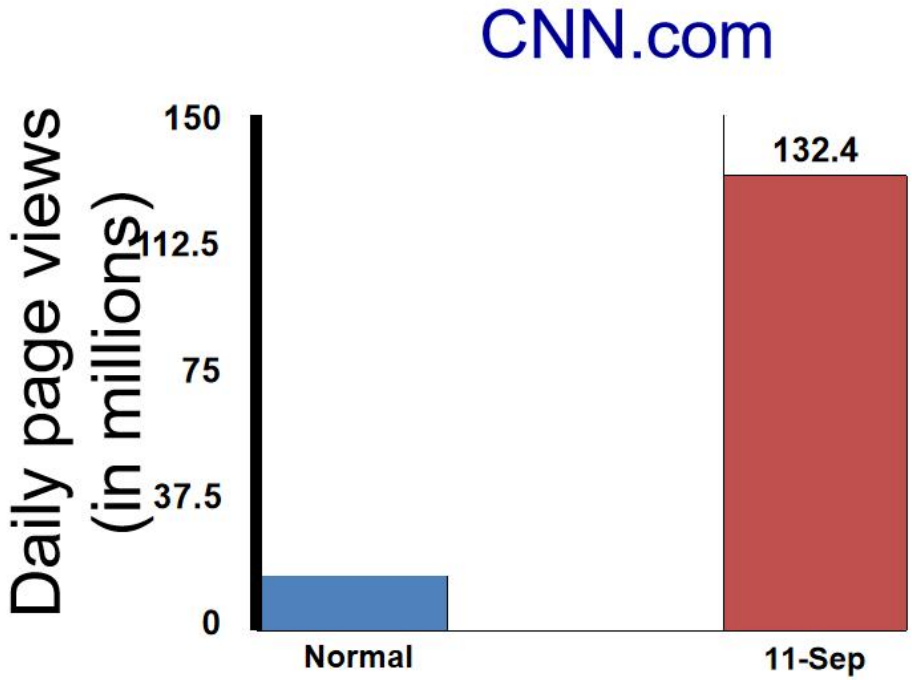
性能度量

- TPS (Transactions Per Second)
  - 客户机在发送请求时开始计时，收到服务器响应后结束计时，以此来计算使用的时间和完成的事务个数，包括了：
  - 1)用户请求服务器
  - 2)服务器自己的内部处理
  - 3)服务器返回给用户
  - 这三个过程，每秒能够完成N个这三个过程，TPS也就是N。
- QPS (Queries Per Second)
  - 每秒能处理查询数目。是一台服务器每秒能够相应的查询次数，是对一个特定的查询服务器在规定时间内所处理流量多少的衡量标准。
  - QPS 基本类似于TPS，但是不同的是，对于一个页面的一次访问，形成一个TPS；但一次页面请求，可能产生多次对服务器的请求，服务器对这些请求，就可计入“QPS”之中。

扩展性Scalability

- Scalability is the ability of a system to handle growing amount of work in a capable manner (适应不断增长的处理任务的能力).
- Scale vertically(垂直扩展)
  - To scale vertically (or scale up) means to add resources to a single node in a system, typically involving the addition of CPUs or memory to a single computer.(提高硬件配置)
- Scale horizontally(水平扩展)
  - To scale horizontally (or scale out) means to add more nodes to a system, such as adding a new computer to a distributed software application.(增加新的计算机分布式计算)

案例1：2001年美国911事件



CNN, NY Times, ABC News
unavailable from 9- 10 AM
(Eastern Time)

案例2：淘宝双十一事件

2010年11月11日，淘宝推出5折优惠的促销活动，2100万人参与，单日成交9.39亿。但是对于这种爆发性流量，超出了淘宝系统的承受能力，被迫停掉“确认收货”业务。

2010年11月11日，某网店旺旺聊天截屏

案例3：“龟速网?” 12306

2012年1月1-7日，“12306”网站日均点击次数已经超过了10亿次,1月9日下午6时许，该网站页面打开速度仍明显慢于其他网站，网站一度无法登录。

“龟速”网站四宗罪，彻底让你没脾气

界面设计丑陋等并不影响购票的问题暂且不谈，人们在使用铁路售票网站12306.CN时发现了以下四宗罪：

第一宗：订票过程繁琐，13道步骤看晕人

根据铁路部门绘制着网购车票流程图，从登录12306网站进行用户注册，到支付成功订票，前后共需要13个步骤。这些步骤足以让对网购不熟悉的人头晕眼花。

第二宗：网络“龟速”，网页时不时崩溃，平均刷新500次才能购到一张票

一步步地输入信息时，购票者时不时就碰到“系统忙”的提示，结果只有不停地重新登录，不停地刷新。有记者根据日点击率和日售票量进行了计算，结果发现平均刷新500次才能够买到一张票。

第三宗：幸运购到票了，结果发现原来“扣钱不出票”

就算足够幸运，“成功”网购火车票，也不一定出票。网络购票“扣钱不出票”情况已出现十多天，至今仍未有效解决。

第四宗：网站还可能存在安全漏洞，有泄密危险

在国内知名的技术论坛CSDN上，一位名叫”特总兵-AK47”的网友研究发现12306网络售票网站存在安全问题，他认为铁道部购票网站可能造成另一次的密码危机。他指出，12306网站技术落后，并且使用明文保存用户信息，很可能有泄密风险

更多案例

- 1999年6月，eBay发生长达22小时当机事件，连带使eBay股价惨跌，五天内跌了将近26%；那一次当机，使公司第二季营收少掉500万美元。
- 2001年1月4日, eBay当机21小时。
- 2005年5月14日，Google当机1小时。
- 2009年8月4日,PayPal全球当机1小时。
- 2010年9月,Facebook当机数小时。
- 2010年12月1日，新浪微博当机将近1小时。
- …

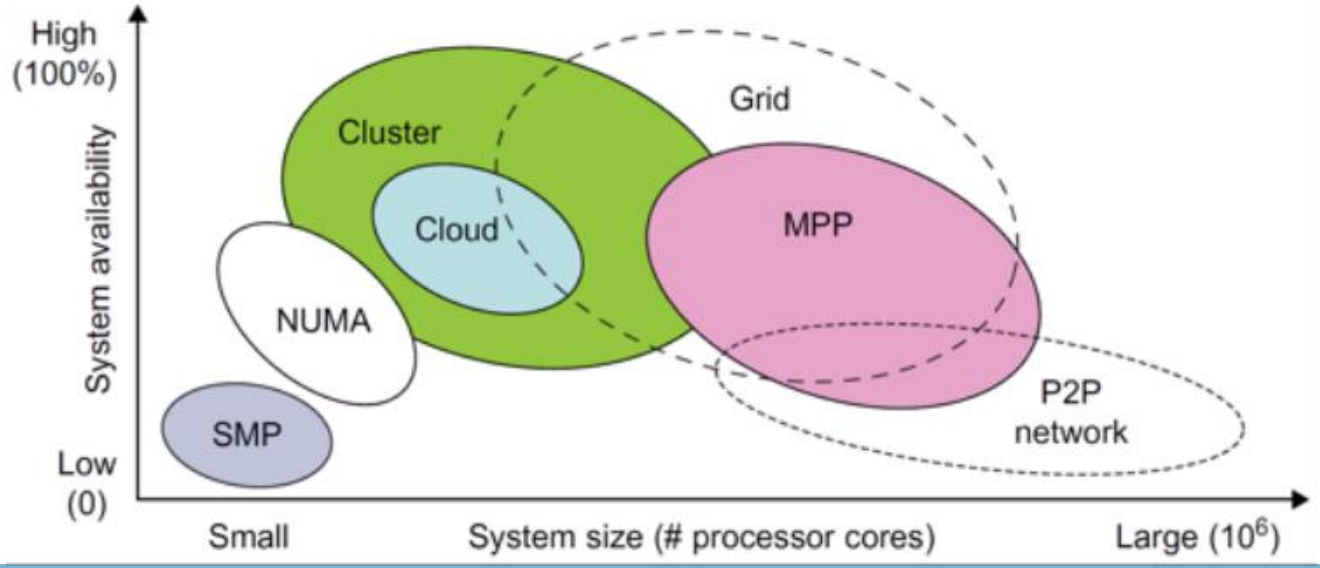
高可用性High availability

- The system has the ability to ensure the its services accessible with reasonable/predictable response times at any time(系统在几乎任何时刻都可被正常访问)
- Availability is usually expressed as a percentage of uptime in a given year (通常量化为一年中正常运行的时间比)

Availability %	Downtime per year	Downtime per month*	Downtime per week
90% ("one nine")	36.5 days	72 hours	16.8 hours
95%	18.25 days	36 hours	8.4 hours
97%	10.96 days	21.6 hours	5.04 hours
98%	7.30 days	14.4 hours	3.36 hours
99% ("two nines")	3.65 days	7.20 hours	1.68 hours
99.5%	1.83 days	3.60 hours	50.4 minutes
99.8%	17.52 hours	86.23 minutes	20.16 minutes
99.9% ("three nines")	8.76 hours	43.2 minutes	10.1 minutes
99.95%	4.38 hours	21.56 minutes	5.04 minutes
99.99% ("four nines")	52.56 minutes	4.32 minutes	1.01 minutes
99.999% ("five nines")	5.26 minutes	25.9 seconds	6.05 seconds
99.9999% ("six nines")	31.5 seconds	2.59 seconds	0.605 seconds

高可用性High availability

- 系统可用性(System Availability)
  - 平均故障时间MTTF Mean Time To Failure
  - 平均修复时间MTTR Mean time to repair
  - 平均故障间隔时间MTBF Mean Time Between Failures = MTTF+MTTR
  - 系统可用性=MTTF/(MTTF+MTTR)



第3章

计算层的软件架构技术

Thanks for listening

涂志莹

哈尔滨工业大学计算机学院

企业与服务计算研究中心