

高性能,高流量,多数据中心
互联网应用架构实战
——以FreeWheel MRM系统后台为例

Diane Yu
Co-Founder, CTO

王迪
FreeWheel核心系统技术总监

2009-04

Design with failure in mind

- There is no bug free software
- There is no failure proof hardware

Partition your data

- Load balance: Even partition
- Easiness to repartition
- Reduce dependency

Redundancy

- Reduce single point of failure
- System auto recoverability

Monitor, monitor, monitor

- Monitor from customer perspective
- Monitor from capacity perspective

KISS

- **Keep: Upkeep**
 - Requirement is meant to change all the time
 - Software is meant to evolve all the time
- **Simple Stupid**
 - More code, more maintenance
 - Something is wrong when it is complicated

Just in time

- **Just in time design**
 - Perfect design doesn't exist
 - Design just enough to get started
- **Just in time refine/refactor**
 - Know when to refactor
 - Refactor in pieces is always better than complete rewrite

服务性能指标

- 广告投放服务Up time 99.99%——每月down <=4分钟
 - 负载均衡
 - Web服务多路转发
 - 前端应用服务器failover设计
 - 多线程
 - Watch Dog/SNMP自动监测
 - Counter服务器，本地缓存+中心同步+后台反馈，避免单点故障
- 单次广告投放延迟 <150ms
 - 轻量级Web Server + FastCGI
 - 广告决策信息提前构建并缓存，减少DB访问，cache分级更新
 - 第三方监测Gomez

高系统吞吐量的设计原则

- 简化前端应用服务逻辑

- 单点服务响应快
- 易于开发/维护
- 易于横向扩展
- 易于提高可靠性

- 减少单个事务处理的状态依赖

- 每次广告请求和确认携带所有必要的决策信息,减少服务器间的通信依赖和请求间的状态依赖。
- 缺点:日志信息冗余,数据量大
- 以空间换时间

- 海量日志处理交由Map-Reduce后台完成

Hadoop是Map-Reduce思想的开源实现

优点:

- 适用于海量信息处理与挖掘
- 开源社区支持，已在大型搜索引擎应用，改进搜索质量
- 可用廉价机器组成大规模集群，良好的自动调度
- 将复杂问题，分解为基本的Mapper和Reducer方法交替完成，编程开发简单。

缺点:

- NFS交换和存储日志数据，I/O负载较高
- 单节点需拥有待处理信息的全部拷贝，处理能力与内存成正比，处理跨节点分散数据的交换时间较高
- 过程中Debug较困难
- 重新处理代价较高

业务待处理日志特点

- 单条日志数据尺寸较大(2k); 数据域较多, 通常在几十; 每日总日志条目数据量较大, G~T级。
- 日志生成时间/空间分散
 - 一次业务事务t由若干关联步骤组成, 如A->B->C, 且同时有若干事务同时发生
 - 各步骤可能发生在不同前端服务器, 如A发生于S1
 - 各步骤均产生一至多条日志, 如A产生A0S1, A1S1
 - 各步骤发生时间分散且间隔不确定, 如A0S1出现在S1的日志S1L1中, A1S1则在S1L2中,
- 日志处理要求归并决策
 - 日志处理过程仅当找到全部有关t的日志, 才可对t进行处理

业务驱动的日志处理需求

- 实时性:每小时, 每天的连续事务处理和数据发布
- 经济性:少量处理节点, 而非大规模廉价集群
- 扩展性:接近业务增长线性吞吐的扩展
- 准确性:数据一致性要求高, 需易于调试和测试
- 重处理:部分日志需要经常重新处理

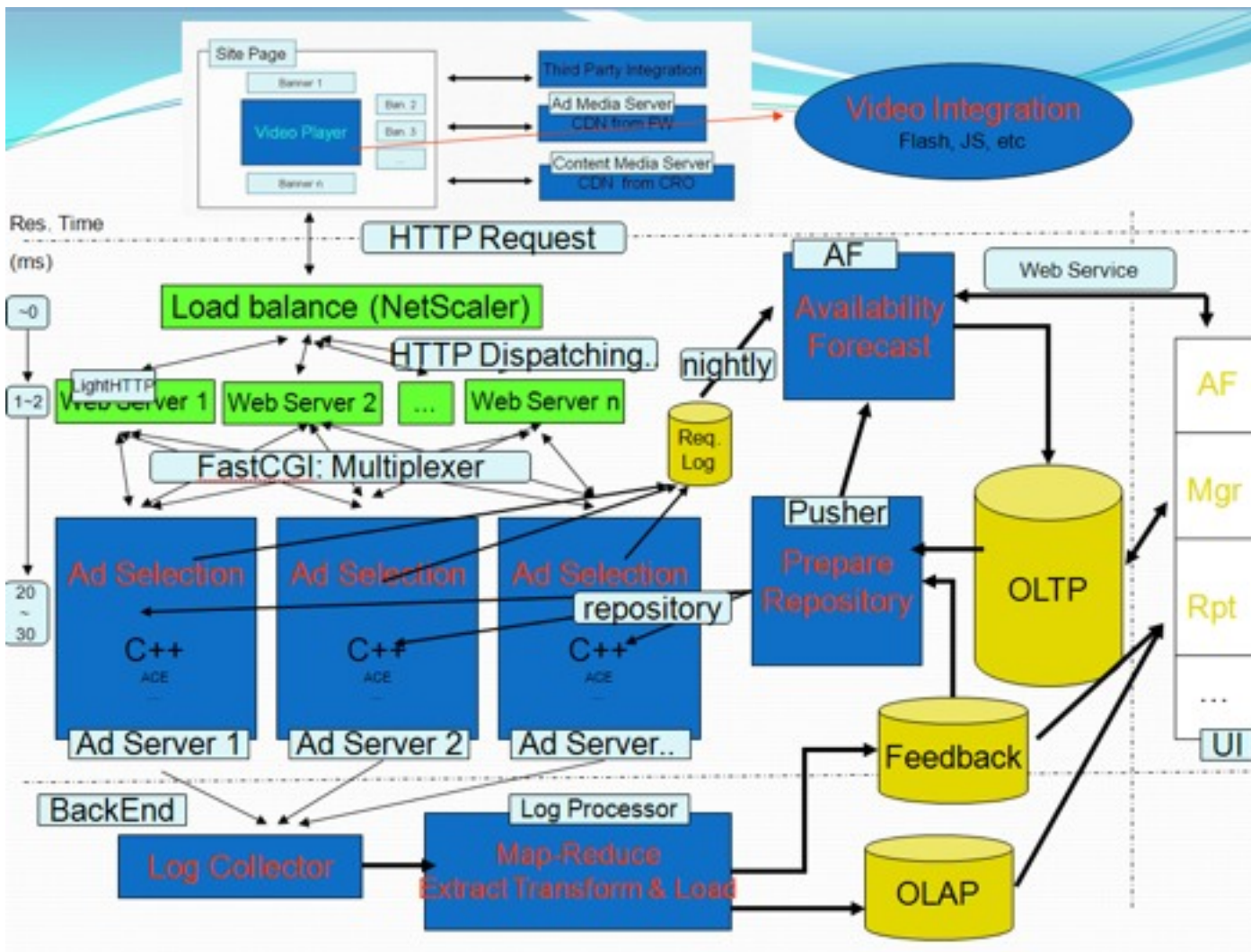
日志处理架构的优化对策:

- 保留Map-Reduce的基本思想, 但放弃Hadoop
- 实现Local Matcher日志预处理, 跨服务器数据预交换: 减少Map-Reduce过程中的跨节点交换

日志处理架构的优化对策：

- 实现Local Reducer日志数据分块：各后台节点只归并处理一组前端服务器日志，将 $O(n)$ 规模问题降低为 $O(n/m)$, m 为节点数，降低了问题复杂度，提高了处理效率。
- 由于“分而治之”，降低了日志重新处理的开销
- 内存使用更加合理，Overhead降低
- 插件式处理子程序，扩展更灵活
- 在Map-Reduce过程中，同时完成ETL

- 多数据中心的好处
 - 更高质量的区域服务体验
 - 负载均衡与故障转移
 - 扩展性考虑
- 跨数据中心的挑战
 - 数据更新的同步和一致性
 - 数据交换负载与稳定性
- 架构设计策略
 - 增量更新,如Master-Slave,Delta Push.
 - 负载均衡IP Sticky
 - 日志局部预处理, 减少数据中心间交换





谢谢各位，欢迎交流！
dwang@freewheel.tv
+86-13901312946