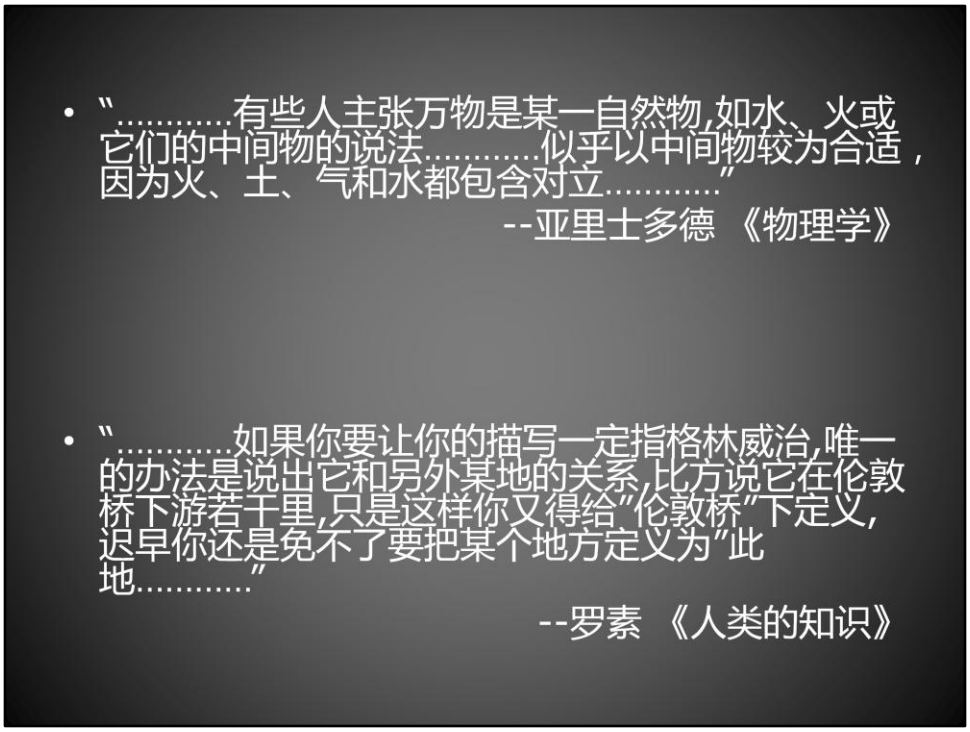
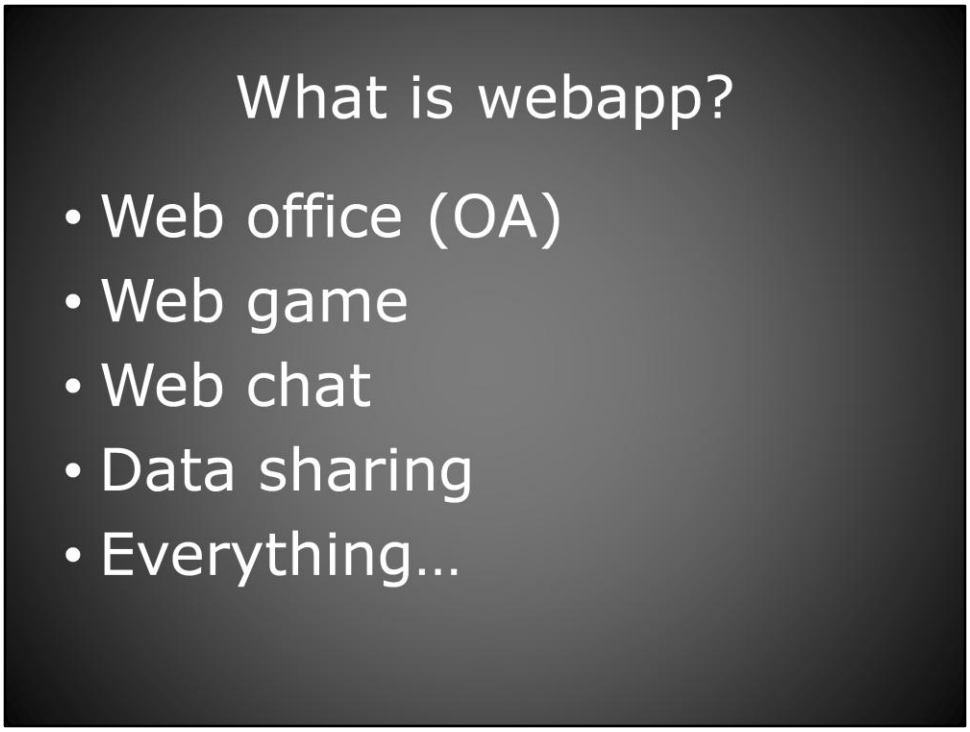
题目是webapp与erlang,主要关注的点为对webapp前景的分析,以及对构建webapp过程中的关键点的理解与分析.并结合一个实例对上述一些观点进行进一步说明.展示如何使用erlang编写一个可扩展的分布式webapp后台.展示如何使用flash编写一个实时的网页客户端,并使之与erlang交互.

FP与近代主流的哲学世界观.



亚里士多德的观点代表了传统编程语言的哲学思想,这种思想直到上个世纪初一直是主流.他认为世界是由”本原(object)”组成的,任何一种事物都可以通过对其它事物的派生,组合来定义.我们对世界的抽象是基于”类型”的.

罗素的例子的背景:
如果让你定义”经度”,就需要定义经度的原点在哪里,没有这个属性的话,”经度”这个类型就无法定义.如果要定义”原点”,必须要说明格林威治是什么.而他认为,无论对格林威治的属性描述得多么全面和完善,总会找到另一个不同的地点与之相同.从人类的观察能力来讲,不引用它与其它事物的关系,是无法定义一个事物的.
罗素的观点代表了更现代的哲学思想,这种思想认为人类没有能力给事物”下定义”,不管对事物属性的描述多么详细,总能找到与这种描述相同的另一种事物.而精确定义一个事物的唯一方式是通过它与其它事物的”联系”.也就是说,世界并不是由类型组成的,它实际上是一系列自恰的”因果关系”,这正是FP思想的基础.



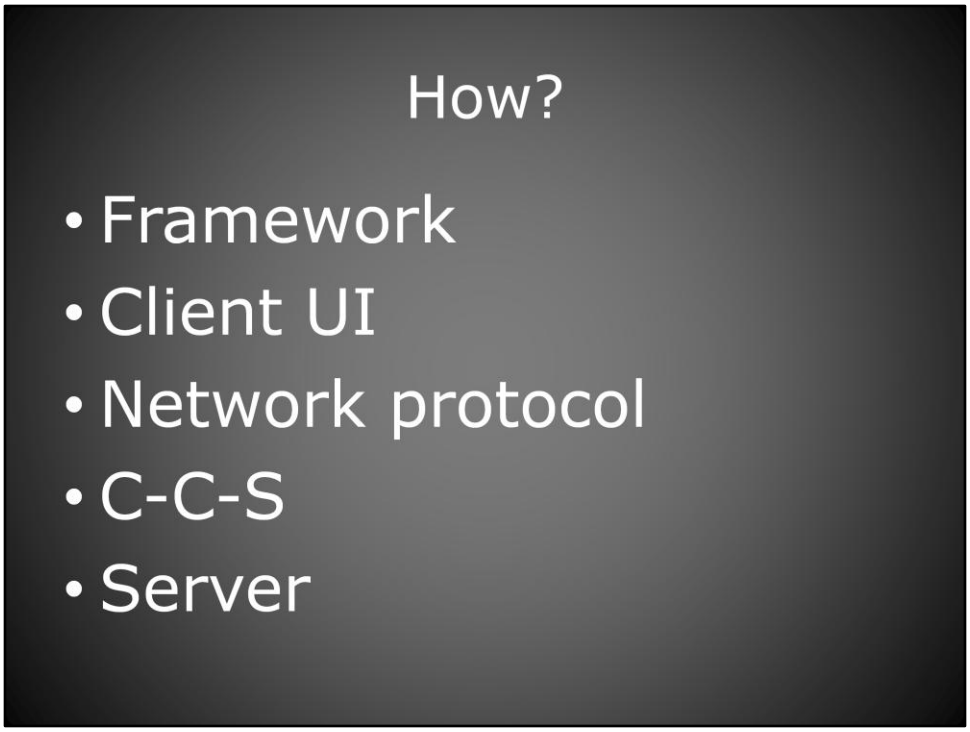
现有webapp的形态,特点,前景.
发挥想象力…

第一个原因是带宽,KB级到MB级的升级让互联网发生了巨大的变化,出现了很多前所未有的应用.而目前有些地区已经有了GB的家庭出口带宽,当网速达到这个量级时,我们会用互联网做些什么?

第二个原因是价格.当带宽价格低于硬件价格时,用户选择低配置的终端而通过网络使用服务端的硬件资源更为划算.

第三个原因是网络终端的变化,对轻便性有很高要求的移动终端,低价,低功耗的瘦终端,它们的出现与普及对webapp有很强的推动作用.

第四个原因是各种维护的成本,这一点对企业应用更为重要.



这一张是对于构建webapp的一些自己的观点.

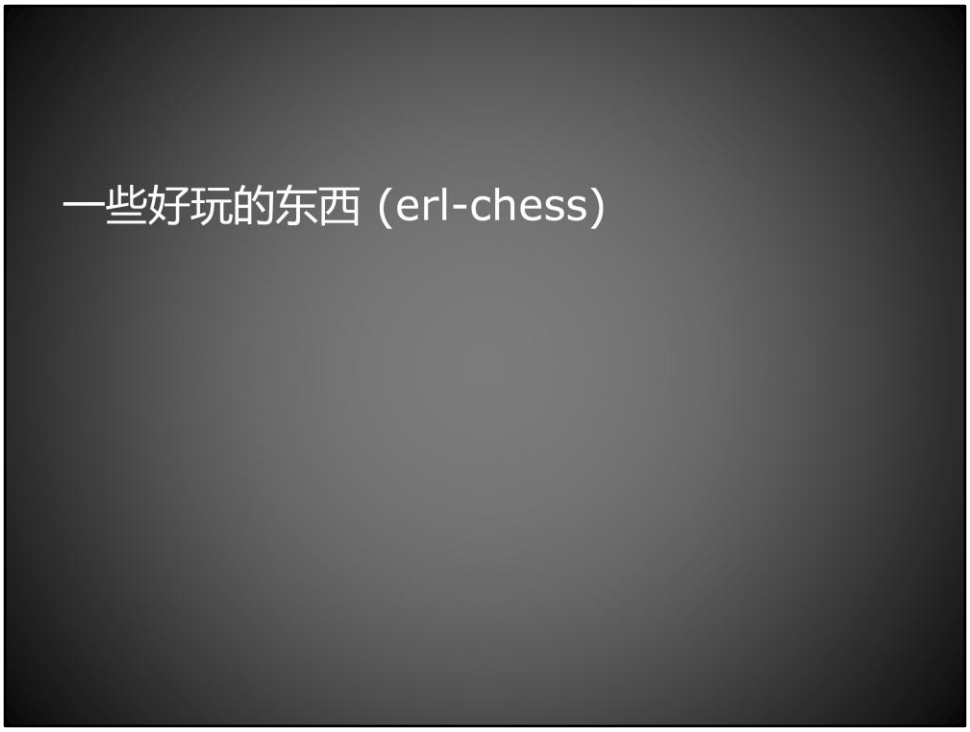
1.现在还没有一个能够使C-S或者B-S架构透明的framework.一个理想化的webapp的framework应该是所谓的web os.
分析一个web os应该是什么样的?(自己开发可以吗?)

2.Html, javascript, flash, silverlight, javafx…(分别说明特色与优缺点.)

3. Comet们能走多远?我们需要的新的协议是什么样的?

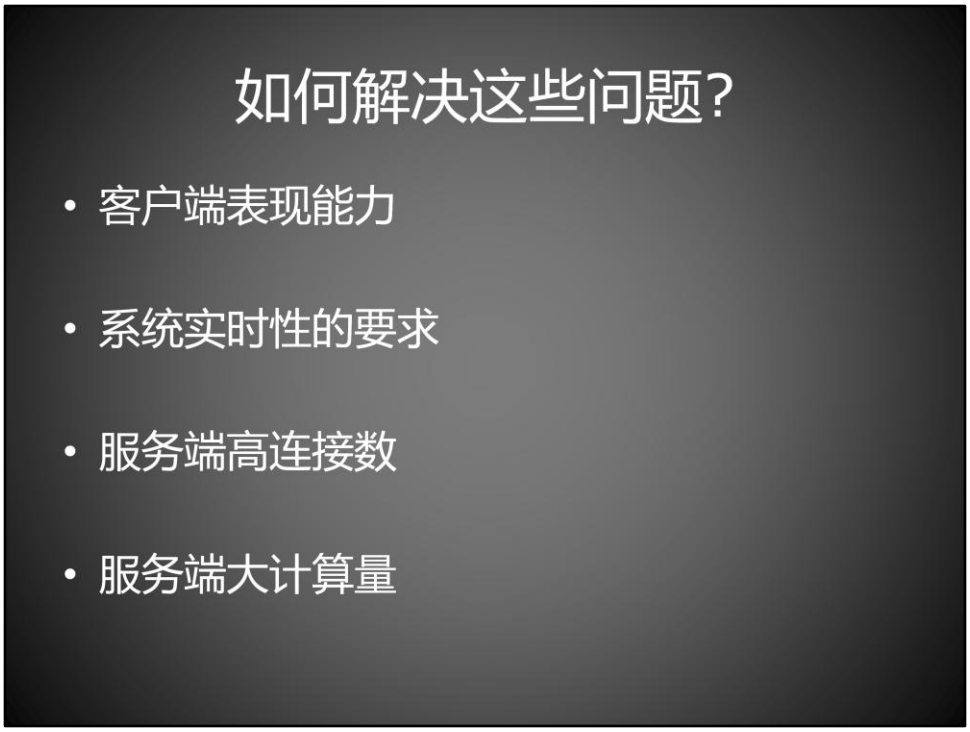
4.C-C-S模型是指在客户端有一个基础的框架(第二个C)负责计算,存储,网络通信等,使得这些问题和第一个C(html, js, flash等)隔离开.(我看好google gears).这种模型对于webapp来说,可能是过渡时期的中庸的解决之道.

5.对于服务端而言,webapp不同于传统http对服务端”轻量级”的访问,所以对服务器连接能力的要求更高(erlang的优势).webapp更复杂的业务逻辑需要的计算和存储能力只能由分布式系统来解决(同样是erlang的优势).

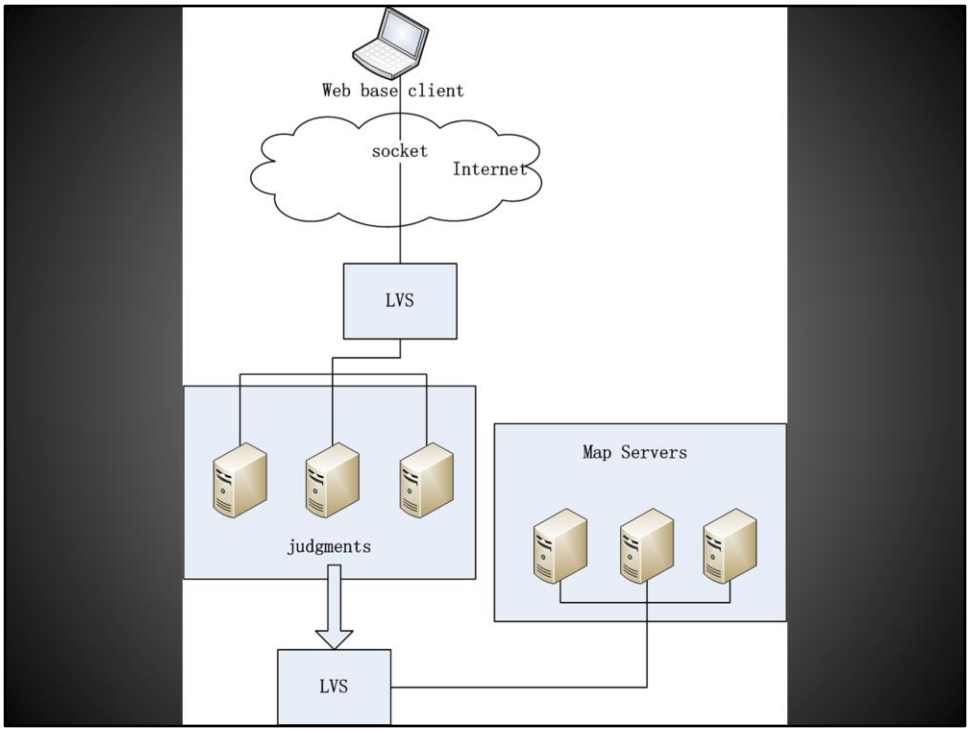


Demo展示

基于上一页提出的一些观点,设计并制作了一个实时的webapp的demo.现在展示一下它的效果.



1.讲解如何使用flash获得更加丰富的表现能力.
2.讲解如何不拘泥于现有协议而达到实时性的要求.
3.讲解如何使用基于LVS的连接负载均衡技术提高后台连接能力.
4.讲解如何采用P2P模型的分布式计算架构,以提高后台计算能力.



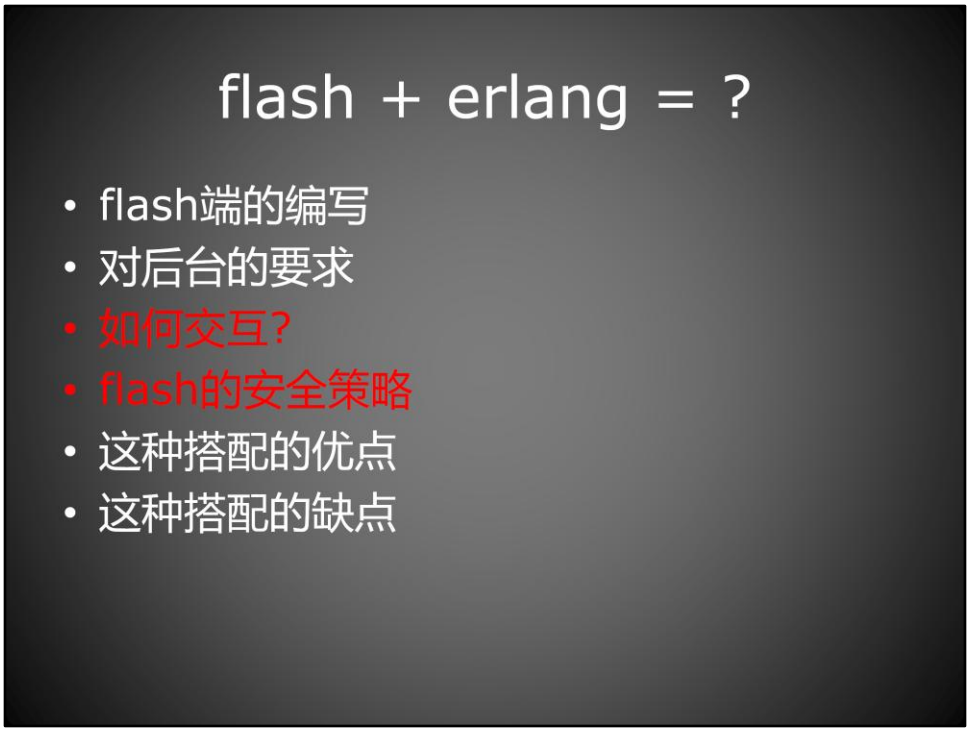
系统结构图

说明:
从宏观上看这个结构与真实世界中的棋类系统是一样的.用户通过一个key(这里的key可以认为是棋盘的标识)去申请裁判,另一个用户如果获得该key,就可以用同样的key去申请裁判(可以是不同的),两人开始后分别由两人的裁判拿key去申请一个棋盘(用于保存实时的数据,记log等).

如何扩容?
在LVS下增加服务器.这是一种高颗粒度的分式布结构.理由:规模 = 小计算量 * 大连接数.

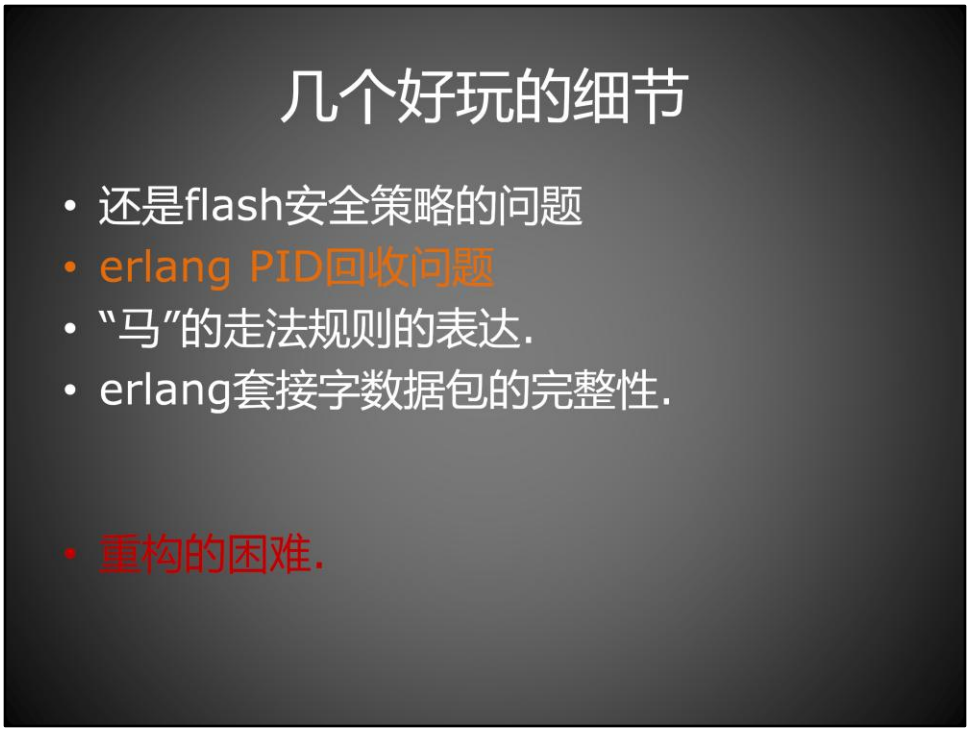
设计原则:
1.尽量把复杂性放在服务端.客户端无状态,只负责显示.
2.协议公开,客户端逻辑不被信任.
3.数据的保存(内存,磁盘)集中式管理,除数据组件外,其它组件全部无状态.

技术要点:
1.客户端技术的选择...
2.网络协议...
3.服务端技术的选择...



这张主要给大家详细讲解下这个系统是如何开发的.以及开发过程中对flash + erlang这种搭配方式的一些体会与见解.

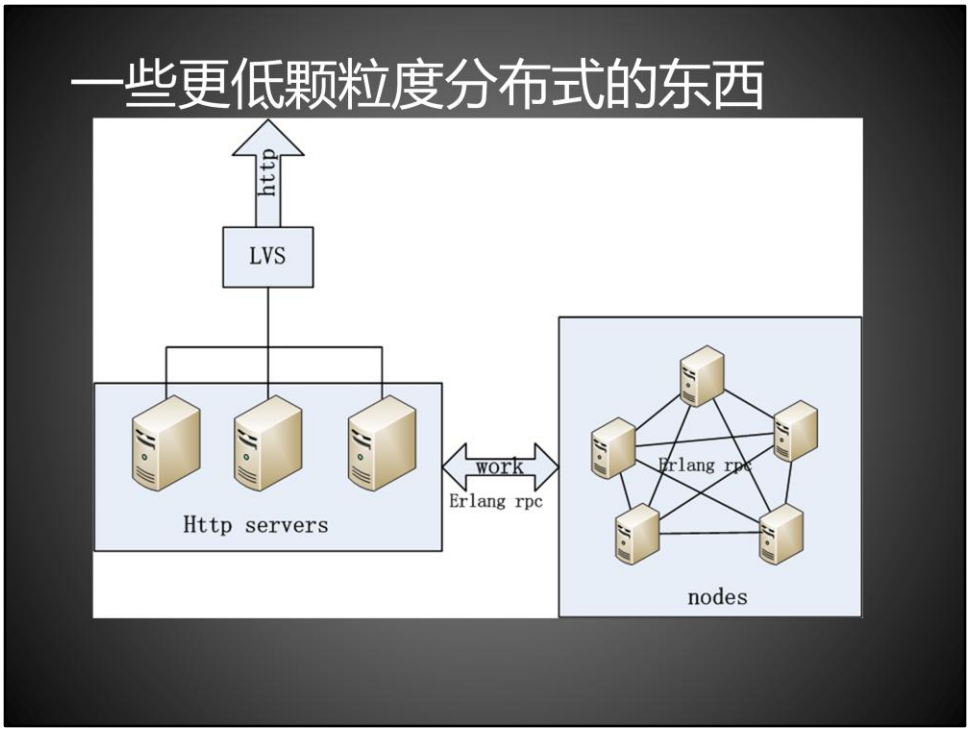
1.简要讲解flash客户端的开发.
2.简要讲解后台erlang socket服务器的开发.
3.讲解交互的方式.(重点)
4.实作过程中遇到的flash安全策略的问题,及解决方法.
5.6.讨论下优缺点.



这张讲几个实作过程中遇到的一些有趣或有价值的细节.

1.Flash会不定时的,"声东击西"的向服务端索要"安全证书".很邪恶~
2.Erlang回收的pid会被迅速的放出.这本身不能算erlang的bug,但是会对调试与排错产生干扰.(实例,bug A与 bug B的关联性).发出10份任务,收回来12份 -__-!
3.体现了erlang优美的语法.
4.Erlang socket使用的细节.

5.讨论下erlang代码重构过程中遇到的困难.



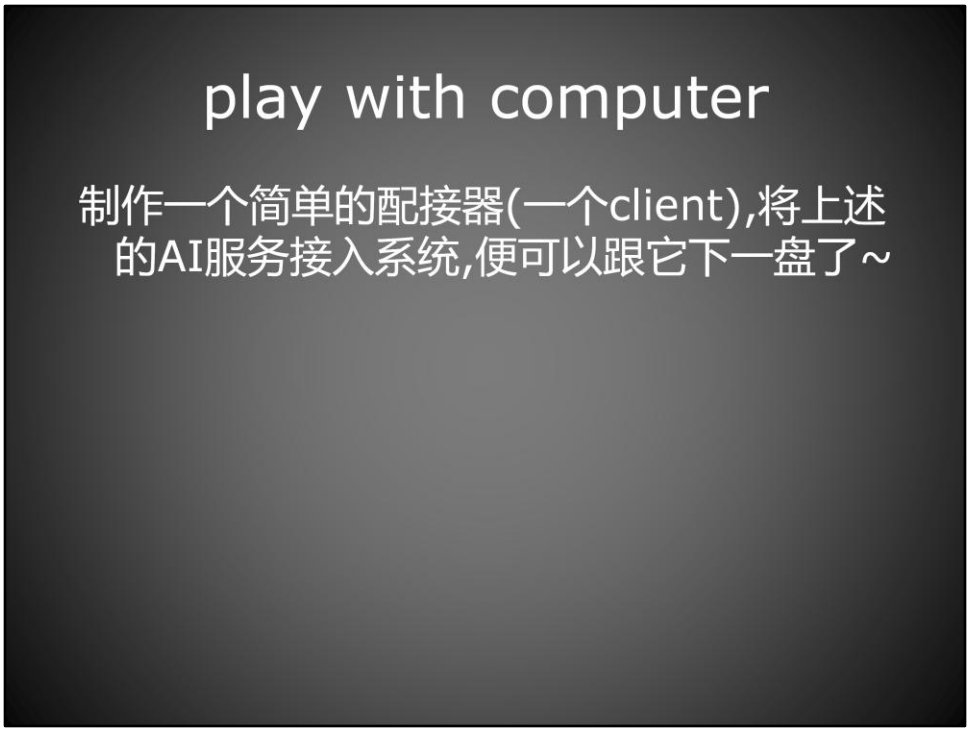
一个象棋机器AI的系统.
为何需要低颗粒度的分布式计算?理由 : 规模 = 大计算量 * 少连接数 or 规模 = 大计算量 * 大连接数

系统结构图
如图所示,整个系统是无状态的,系统接受一个棋盘局势,并给出系统认为的一步最佳走法.此过程为对该系统的一次访问.基于这个原因,接口协议选择http

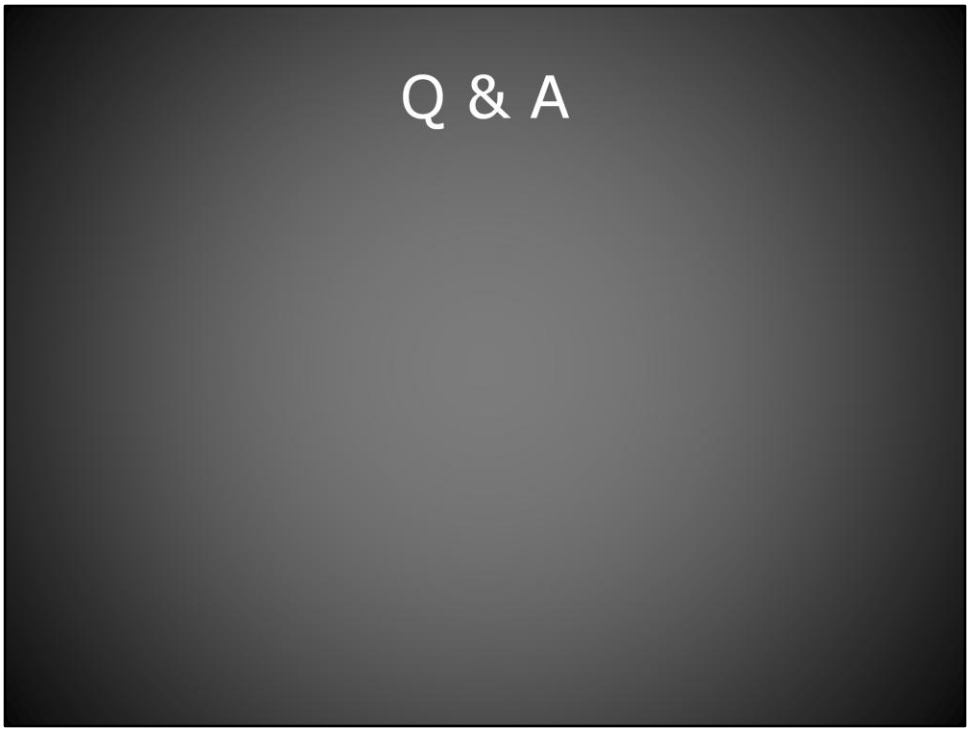
http server负责处理http请求,并生成work,通过erlang rpc传递给一个计算结点(随机选择).

这里使用的AI算法可以归结为对一棵棋盘局势树每个枝叶的评估与结果的汇总.故每个结点的计算也是无状态的.所有结点组成一个P2P网络,当一个结点认为自己的能力无法快速处理某个局势时,就会把该局势的一些子局势发给其它结点处理,并将结果汇总,报告给它的上一级.报告到最上一级时,该结果就可以返回给http server了.当某个结点无法快速返回正确结果时,它的父结点会把该份work分配给其它结点处理.

如何扩容?
增加http server可以扩容连接能力,(动态)增加node可以扩容计算能力.可以动态的原因是还有一个name server的服务器(图中未画出),负责动态管理结点的列表,状态,以及生命周期等.



Demo的AI部分的展示



Q&A