我在eBay的两年

前几天公司年会，里面有个环节，邀请在公司待满10年的员工上台，颁发奖杯。想到台上的孩爸孩妈，当年也是像我一样的毕业没多久的新人，已经为公司奉献了10年青春，很是煽情。在这即将入职两周年之际，写一篇总结性的文章，等到我的10年，再读起来的时候，一定是件不错的事情。

作为没有多少写作功底的人，记流水账是我最拿手的。本文后面的章节，会按照时间顺序，罗列一下我的经历。如果其中刚好有对大家的启发，那就太赞了。

关于Hadoop工作原理的一次分享

拿到offer之后，入职之前，我在组里实习过一段时间。Kevin是我的mentor. 当时得知后面会做hadoop相关的开发。
我的实验室是网络方向，导师是做基础研究的，重理论，在学校的时候，用过storm（真的只是简单的用过），对hadoop只是知道名字而已，好慌。Kevin让我先熟悉一下，回头分享给大家。
不得不说，网上和组里关于hadoop的资料很多，讲的深浅不一样，熟悉这段知识的时候，有种越看，不知道的越多的感觉。一旦开始深入了解一个系统，就会想一些细节为什么要这么设计，某个模块到底是怎么工作的。想到要给大家讲，就逼着自己整理的尽量条例，能挖的细节，就挖下去。
看了两个星期左右吧，给组里的同事做了一次sharing。当天几乎所有的人都来了。。慌慌张张的，好多东西因为理解的不够，缺乏实践，根本讲不清楚。好在，这段时间看的资料，激发了应该去读一些源码的兴趣，积累下来的知识，对我后面参与pulsar feed，起到了关键的作用。
我意识到，

- 验证对某个系统理解程度的最好方式，就是尝试着讲出来。你会逼着自己深入下去的。
- 新人可以尝试着站在新人的角度，写一些新手教学之类的，既可以总结，又可以帮助后来人。

Pulsar feed 2.0

Pulsar feed 2.0 是我参与的第一个正式项目。它的主要功能是，从HDFS load 最近几个小时的raw events，将同一个GUID的一组group成session，并从中识别出由robot产生的，分folder的写回HDFS。Kevin当时已对已有的系统做好了重构，整个过程被切分成几个子job。第一阶段，会load raw数据，生成session和bot detection所需要的metrics，这部分逻辑是Map Reduce实现的。第二个阶段，基于metrics和bot rule，识别bot，这部分逻辑是Spark实现的。第三个阶段，输出最终结果，基于MapReduce实现。验证job运行结果的方式，是跟另一个team的输出结果做对照，看bot的覆盖率相差

多少。
我的工作是在Kevin所写框架的基础上，移植老系统里的跟第一和第三阶段相关的逻辑。人生中第一次意识到，大数据，不是论文中的一个单词，而是实实在在的。24小时的raw数据有2-3个T，我们的job需要花7-8个小时去处理。有好几次，因为写的bug，会导致前一天跑出来的结果，全部作废。快release的那段时间，经常凌晨两三点修bug，然后提交job，第二天9点起来看结果。
中间还出过一次事情。有天我改了逻辑之后，直接替换掉了线上的jar包，因为bot rule之间的关联性，整体的结果下降了几十倍。我没有保留上一次的运行环境，所以没法roll back。天天担心被炒。。我跟晓菊白天黑夜的查root cause，最后发现是某个metric在我修改完之后变了，导致其中一个bot rule抢 占了其他bot rule的events。
老板Nemo并没有因为这事责怪我。他说，人总会犯错，只要犯错之后能积极的查原因，并从制度的角度思考，怎样能避免犯同样的错，就是好的。当时只是很感动（庆幸不用收拾行李滚蛋），直到很后来才知道，这是一种culture，叫blameless postmortem.
虽然从结果来看，这个项目不能称之为特别成功，但是确实凝聚了很多人的心血。我从Kevin那里见识了什么叫架构，什么叫performance tuning。晓菊和我从那时候开始，成为组里最早的加班二人组，建立了牢固的革命友谊。
我意识到，

- 大型项目里有复杂的逻辑，充分测试之前，不要盲目的部署新的代码
- 不要人工部署PROD环境，即使不能使用现有的release的流程，也要写脚本来做
- 新人不要怕犯错，但是要避免重复犯同一个错

Large session detection

后来Kevin去做messaging，Yanger变成了我和晓菊的mentor。我们的工作是优化pulsar feed 2.0的process，尽量的automation，减少人力的support。
从eagle上输出的数据来看，2.0第一个阶段的job运行时间很不稳定，有一段时间，甚至需要7-8个小时去处理一个小时的数据。从数据量来估计，应该20分钟左右就能结束的。我们去MapReduce的console上看一个job下面的task运行时间，发现所有的mapper task结束的时间很均匀，但是个别reduer会花费几个小时，以致拖慢了整个job结束的时间。
应该是data skew，我们意识到。从mapper到reducer，根据的是guid做hash。难道是某个guid的event特别多？我们在code里面写了更多的log，验证了猜想。有一些guid会有超过10万个event，并 且一直连续的产生，造成一直没有在sessionization中被输出，拿到这一组events的reducer就会跑的 特别慢。
有很多event的session在我们的概念中被称作large session。large session是测试或者由其他bot产生，对数据分析没有用处的，应该滤掉。之前，我们有一个配置文件，写死了有哪些guid是来自large session。但是从某个时间点开始，有新的large session guid动态产生。

我们需要动态的维护这个配置文件，识别出large session的guid，及时的更新配置未见，在尽量早的时候drop掉。Yanger和我做了很多实验验证性工作，明确的定义出问题，并且设计了一个分布式的LRU black list。后来给Nemo review的时候，他建议我们尝试去跟其他系统集成，直接输出哪些guid是由什么team生成的，以此提醒他们是否是程序中出现了什么问题。
上线之后，sessionization的运行时间稳定的在20分钟之内，performance提升了35%。我后来写了一篇学院派的tech blog，详细的记录了整套设计。
我意识到，

- 发现问题和定义问题总是比解决问题更重要，也更难
- 在公司也会有研究性的工作，而且可以很cool

Classified, UBI processing, Krylov, KT，schema registry与迷之30s

后来公司有org change，组里下一步要做的事情没有完全确定下来，是宝贵的充电阶段，也趁机接触了很多不同的项目。
Pulsar feed2.0 稳定之后，有很多小部门的use case，统称为classified。我们作为一个platform，接收上游过来的数据，做好sessionization，分类的输出到HDFS。
接下来是尝试redesign UBI processing，将pulsar feed2.0的设计思想融入进去。
还参与过krylov，是一个machine learning的platform，我们最早计划做成像microsoft所提供的那样，支持很多算法和modle，用户通过拖动，自定义的处理自己的data set。Xinglang带着我们几个，做出了poc，定义了很多metadata。可惜项目最后由别的组take，方向好像也和我们当时的打算相差很多。
再后来组里要接手从美国某个team移交过来的一堆系统，天天跟他们开会，听他们讲做的东西。我读了一些schema registry相关的文档，学了他的设计，并且搭建了我们自己的cluster。StraaS负责从openstack申请机器，自动化配置，形成cluster。当时觉得，straas是所有系统里最复杂的一个，有好多好多子模块，用了各种各样的语言和脚本。所有跟StraaS相关的KT都是在听天书。。
StraaS有个功能，叫guestcmd，用户给straas service发一个request，straas会把request转到相应的node上面，运行并返回结果。Messaging team需要通过call这个api，来创建kafka的topic或者运行其他跟kafka相关的指令。
问题在于，每次call guestcmd都需要等待1分钟左右才能返回结果，即使只是简单的list一下topic。Kevin他们对此相当的头疼，问美国team，他们表示这确实是个问题，他们好久之前就发现了，但guestcmd本来就是个后门，不是主体的功能，在尝试修复并失败过几次之后，选择耐心等待。guestcmd是messaing team必须要依赖的功能，1分钟的延时是完全影响体验的。
我对performance tuning很感兴趣，迷茫了一段时间之后，终于有一个很具挑战性的问题了。但是我对straas的了解仅限于怎么搭建一个private的server，模块的功能了解，python的code能看懂，自

己没写过。
为了找到root cause，我开始深入的读straas的code，搭建了一个私服，在我怀疑会block的地方打log，发现每个guestcmd会由两次rpc调用，从API到tm一次，从tm到目标node一次，每次会花30s，Yanger跟我给这种现象命了名，代号”迷之30s“。
再深入的，straas的RPC使用的是openstack提供的oslo.messaging。要破解这个难题，得理解oslo.messaging的工作原理。我们去读了相关的博客，有了基础的背景知识。python是一种无类型的脚本语 言，无需提前编译打包，变量使用之前也无需声明，这给我们的调试带来了极大的困难。不知哪来的变 量，也不知道他是什么类型。还好，偶然间我们发现了一个窍门。用print命令，可以直接打印出对象的 类路径。我们开始往oslo.messaging包hack代码，打更多的log出来。
两天之后，问题的根源被揪出来了。发起RPC之前，需要先声明一个transport对象，用于对传输数据通路的抽象，我们的代码中每次rpc都会声明这个对象，用完之后cleanup掉，正是在cleanup的时候，一个while true的pull thread会等待30s的超时。
解决的办法简单的可怕，transport对象只需要全局的声明一次，所有的RPC client都加载它。只有当整个解释器需要退出的时候，才call一次。一次RPC的时间从30s骤降到几十ms。
”迷之30s“从某种意义上宣称了中国区有能力接管straas。我作为参与者之一，有幸第一次给GM Vivian做了15分钟的presentation，她最后笑着说，”这是典型的拧一个螺丝，收2万的例子啊“。从这个case 中，我熟悉了straas和python，一个想法在我脑海中诞生了。
我意识到，

- 在公司里，唯一不变的就是变化。大胆的拥抱变化，才利于自己的成长
- 源码之前，了无秘密。想要理解一个系统，读他的code吧
- 机会不会重复的到来，所以，该抓住的时候，就咬着牙加油干吧

重构StraaS

旧的straas是个很烂的项目，这是众所周知的。作为RHEOS的底层，Nemo对此耿耿于怀。K8在这个时候开始慢慢兴起，公司里有tess team，基于K8提供一些公司specific的功能和强化。要不要弃掉现在的这坨垃圾，基于K8重新build一套新的mini paas呢?
一次one on one meeting上，我跟Nemo说我想重构straas。在迷之30s的case中，我通读了关键代码，发现它的结构有很大的问题，配置管理乱起八糟，大量重复冗余的code。但是，在tool的选取和部 分细节的实现上，有很多可取之处。所以基于其上，做更多的抽象，复用细节代码，可以在几个月之内 完成优化。
非常意外的，Nemo立刻就支持了我的想法，即使我连一个具体的设计都没提出来。”去搞吧，我相信你。“
我像打了鸡血，开始没日没夜的做设计，跟Ken，Yanger和XuXin讨论细节。大年三十的晚上，还在调试code，做POC。过年之后，总体的plan被敲定，是时候大干一场了。

之后的几个月，我们先后release了0.7.4， 1.0.0和1.0.1三个版本。第一个版本，支持了cname和zookeeper的replacement，将在此之前替换一台有问题的zookeeper，需要2个多小时的人工support，降低到现在只要一个request，5分钟的全自动。1.0.0完成了总体的重构，以前想改动一个配置，或者添加一个新的component，几乎需要修改所有的module，并做一次长达两个星期的release，现在只需要编写实现相关的脚本，改动一下配置就好了。第三个版本提升了总体的稳定性，在automation程度上更高。
作为RHEOS的最底层，我们是其他子项目的dependency，为了RHEOS总体的timeline，整个小team拼了老命的delivery，总算赶上了。
我意识到，

- 知遇之恩，不仅是古装剧中的一句台词
- 说一个项目烂，要给出充分的理由，如果能朽木开花，就会很赞

Poseidon

交付了承诺的功能之后，我离开了straas team。RHEOS进入了稳定阶段，开始着手opensource。StraaS作为其中独立的模块，比较适合先行一步。我开始对straas第二次重构，从中剥离出eBay specific的模块，将IaaS层抽象出来，以支持其他的cloud service，比如AWS。workflow和configuration management重写了很多逻辑，以支持更广泛的扩展性和更加generic的use case。
两个月左右，straas的主体功能被移植到AWS，相比于对内的version，它更加的纯粹而灵活。从scope来说，它会用于manage所有跟stream相关的cluster，比如zookeeper，storm和kafka。我们起了一个霸气的名字，圣斗士里的海王“Poseidon”。
由于优先级的调动，这个项目后来被搁置了。静静的等待复活。
我意识到，

- 对于做技术的人来说，最高的礼赞莫过于，“你的项目可以opensource”
- 最佳的design是相对的，解决实际的问题时，总会有trade off，所以在资源条件限制下，要容忍一定的不完美

Messaging

Poseidon跑起来之后，我的组织关系发生了一点小变动，Mentor变成了Jianbo。小组从streaming变成了Messaging。
Jianbo刚来的时候，我俩天天去功夫食府吃白菜酸汤饺子，有时候还分着吃，有深厚的革命友谊。所以mentor的变动并没有太大的影响。真正有影响的是scope。

这次scope变动对我来说是个不小的挑战，离开熟悉的知识圈，也第一次从零开始架构一个大型系统。直到今天，我对完成这次挑战也没有十足的把握。好在，Nemo和Jianbo一直对我非常支持。
Messaging最核心的产品是，messaging as a service(MaaS)。它会定义一组标准的messaing语义，屏蔽掉底层的messaing provider，对外暴露restful和RPC的interface。基于之上，我们准备逐步的淘汰掉现在的BES（一个存在了10多年的老系统，底层是oracle的messaing系统）和notification platform（基于BES之上，支持push语义）。
9月底的时候，去美国出差，给Sami review我们的设计和API，他很感兴趣，并从design的角度，给了我们很多实质性的建议。最关键的，一个全新的，充满爱与关怀的名字”NuMessage". 那是我第一次跟VP做presentation，紧张的声音发抖，要不是Kevin一直帮助我解释，估计当场抽过去了。
我们的架构师Connie对NuMessage也非常的关注，曾经兴致勃勃的给我讲过几个小时她的想法。我们吸收了她的许多意见和建议，NuMessage正是有了她的帮助，才有了今天的样子。
后来我写了很多详细的文档，介绍我们的设计和计划。做了几个POC，证明我们的想法可行。跟Jianbo一起定了总体的计划和每一个小的release的scope。搭建好程序框架之后，整个项目启动了起来。
期间比较有意思的一件事情，为了支持straas的case，我们需要提供一个oslo.messaing的driver，这是第一个version中最具挑战的部分，我们在计划的时候，定了需要45天的时间来完成。后来真正做的时 候，发现我在神秘30s的case中看的东西，好多都可以用得上，在花了10天左右理清脉络之后，只花了 两天的开发时间，就跑了起来。有一种冥冥中自有天命的错觉。。
因为还在进行中，很多事情还没有定论，所以对messaing的介绍就到此为止了。
我意识到，

- 跳出舒适圈真的不舒适，但如果这是成长的必经之路的话，我选择接受
- 没有没用的知识，没准哪一天它会发挥奇妙的作用

Skunkworks

两年之间，我参见了两次skunkworks，项目两年都被选中去美国参见过EXPO，两年最终都没有拿到大奖。。
第一年的题目跟search有关，我们尝试让用户来自己输入preference，微调search返回list的排序结果。这个想法受到了专业人士非常强烈的challenge，但试用过的用户却觉得是个不错的尝试。Tao哥因此上了一次HUB首页。我们最终没有得奖，再一次有力证明了他的颜值，真的可以当饭吃。
第二年的题目跟product template有关，idea本身来自Nemo。Basic idea是依赖于eBay上的用户来贡献和review product template，是crowd sourcing的应用。代码量很大，使用了非常高端的SVM，并由著名架构师Tang架亲自设计，是Yang经理继pulsar bot之后第一次出山coding，Edward临危受命，独自扛起UI的大旗，是Basu java7天速成效果的有力证明。虽然拿到了Hackweek的第三名，最终还是没能打动Structured data的VP。哎，是命运。

Skunkworks真的很有意思，一帮子人在一个小屋里，一边吹牛，一边coding。写完一个功能之后，感觉大奖已经稳稳的入手的那种激动心情，是不亚于真正收获了大奖的。
不多说，今年我们还会参加，并且不卖关子，内容跟AI有关。

最后的一点总结

今天的我，虽然完全不能称之为成功，但过得很开心。工作之后要想开心，并不简单，因为首先有几个条件：

- 一个好老板：好，不是为人Nice，而是非常清楚整个team的目标和方向，以能力来用人，并且愿意为手下的人争取应得的利益。
- 对做的事情感兴趣：计算机整个行业，分支多的可怕，要想开心，一定要做你感兴趣的方向。如果现在还没有，那么不防多接触一点。
- 适合自己的文化：每个公司都有自己的文化，如果你的风格刚好和公司或团队的文化切合，那么恭喜你。对我来说，eBay的文化是自由而开放的。每个人都可以发表自己的意见，每个人的想法都会被尊重。

满足了所有的条件，那么离开心只剩一步之遥了，而且是完全可以做到的：努力。
曾经有一位前辈跟我说过，毕业前两年的work life balance就是只有工作，没有生活，因为以后能达到的高度，决定于最初几年知识积累的厚度。我非常庆幸自己能在毕业之前就听到这番话，也正在慢慢开始体会到它真正的价值。
路还很长，希望自己对技术的热情，永不熄灭。