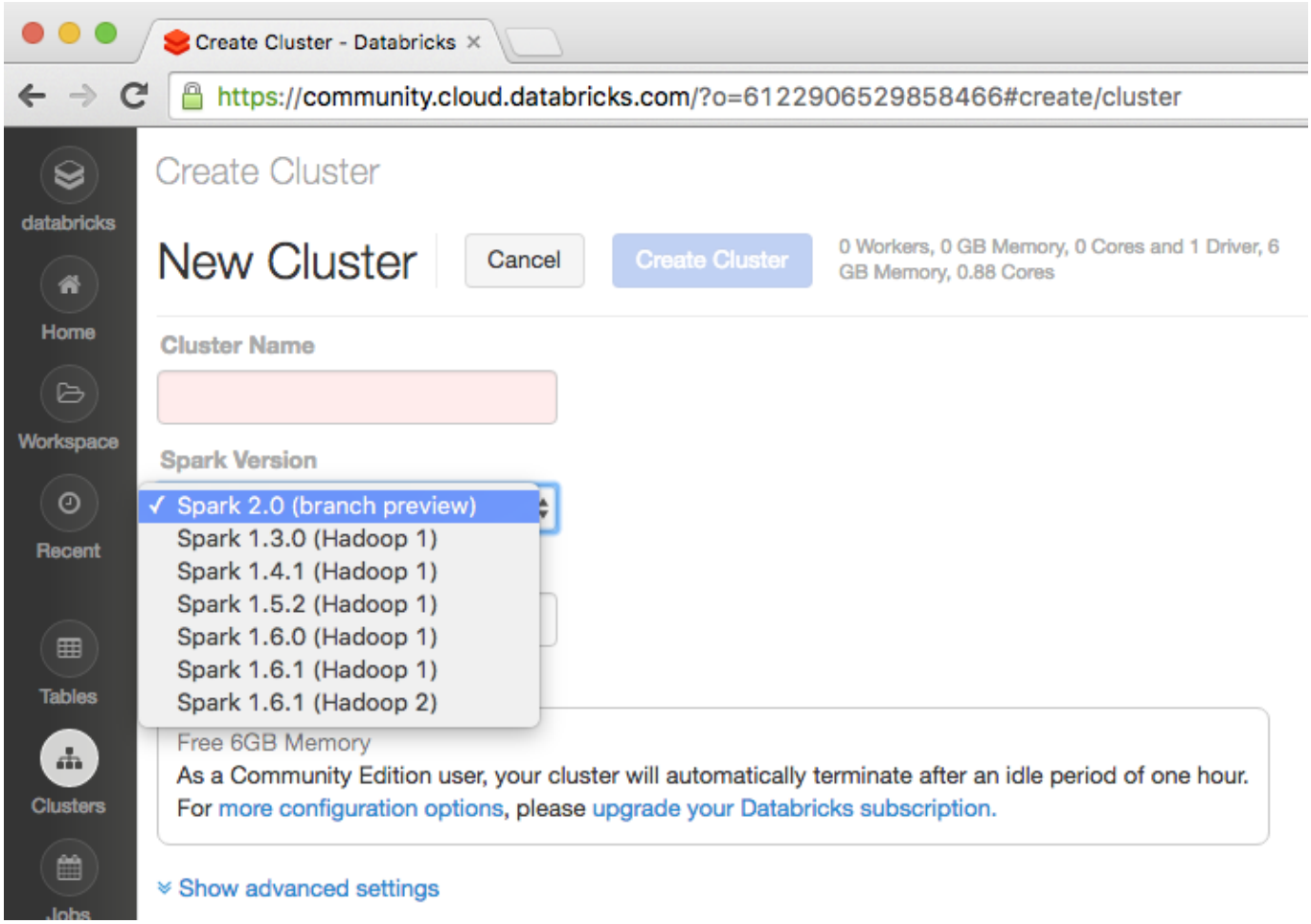
Spark 2.0技术预览：更容易、更快速、更智能

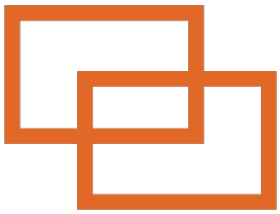
在过去的几个月时间里，我们一直忙于我们所爱的大数据开源软件的下一个主要版本开发工作：Apache Spark 2.0。Spark 1.0已经出现了2年时间，在此期间，我们听到了赞美以及投诉。Spark 2.0的开发基于我们过去两年学到的：用户所喜爱的我们加倍投入；用户抱怨的我们努力提高。本文将总结Spark 2.0的三大主题：更容易、更快速、更智能。更深入的介绍将会在后面博客进行介绍。

我们很高兴地宣布Apache Spark 2.0技术预览今天就可以在Databricks Community Edition中看到，该预览版本是构建在branch-2.0基础上。当启动了集群之后，我们可以简单地选择Spark 2.0 (branch preview)来使用这个预览版，如下所示：



然而最终版的Apache Spark 2.0发行将会在几个星期之后，本技术预览版的目的是基于branch-2.0上提供可以访问Spark 2.0功能。通过这种方式，你可以满足你的好奇心；而且我们可以在发行最终版的Spark 2.0之前就可以获取到用户的反馈和Bug报告。

现在让我们来看看Spark 2.0最新的进展：



更容易的SQL和Streamlined APIs

Spark 2.0主要聚焦于两个方面：（1）、对标准的SQL支持（2）、统一DataFrame和Dataset API。

在SQL方面，Spark 2.0已经显著地扩大了它的SQL功能，比如引进了一个新的ANSI SQL解析器和对子查询的支持。现在Spark 2.0已经可以运行TPC-DS所有的99个查询，这99个查询需要SQL 2003的许多特性。因为SQL是Spark应用程序的主要接口之一，Spark 2.0 SQL的扩展大幅减少了应用程序往Spark迁移的代价。

在编程API方面，我们对API进行了精简。

1、统一Scala和Java中DataFrames和Datasets的API：从Spark 2.0开始，DataFrame仅仅是Dataset的一个别名。有类型的方法(typed methods)（比如：map, filter, groupByKey）和无类型的方法(untyped methods)(比如：select, groupBy)目前在Dataset类上可用。同样，新的Dataset接口也在Structured Streaming中使用。因为编译时类型安全(compile-time type-safety)在Python和R中并不是语言特性，所以Dataset的概念并不在这些语言中提供相应的API。而DataFrame仍然作为这些语言的主要编程抽象。

2、SparkSession：一个新的切入点，用于替代旧的SQLContext和HiveContext。对于那些使用DataFrame API的用户，一个常见的困惑就是我们正在使用哪个context？现在我们可以使用SparkSession了，其涵括了SQLContext和HiveContext，仅仅提供一个切入点。需要注意的是为了向后兼容，旧的SQLContext和HiveContext目前仍然可以使用。

3、简单以及性能更好的Accumulator API：Spark 2.0中设计出一种新的Accumulator API，它拥有更加简洁的类型层次，而且支持基本类型。为了向后兼容，旧的Accumulator API仍然可以使用。

4、基于DataFrame的Machine Learning API可以作为主要的ML API了：在Spark 2.0中，spark.ml包以其pipeline API将会作为主要的机器学习API了，而之前的spark.mllib仍然会保存，将来的开发会聚集在基于DataFrame的API上。

5、Machine learning
pipeline持久化：现在用户可以保存和加载Spark支持所有语言的Machine learning pipeline和models。

6、R的分布式算法：在R语言中添加支持了Generalized Linear Models (GLM), Naive Bayes, Survival Regression, and K-Means。

更快：Spark作为编译器

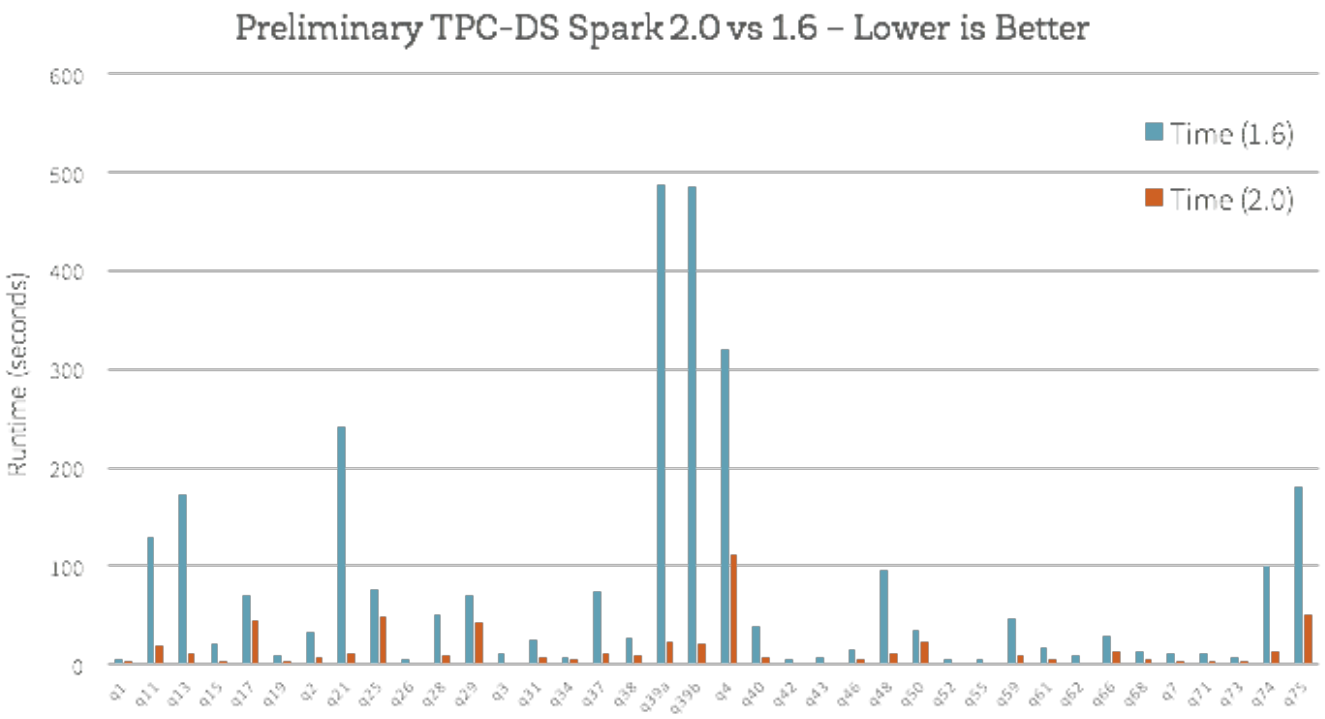
根据以往的调查，91%的用户认为Spark的最重要的方面就是性能，结果性能优化在Spark开发中都会看的比较重。

Spark 2.0中附带了第二代Tungsten engine，这一代引擎是建立在现代编译器和MPP数据库的想法上，并且把它们应用于数据的处理过程中。主要想法是通过在运行期间优化那些拖慢整个查询的代码到一个单独的函数中，消除虚拟函数的调用以及利用CPU寄存器来存放那些中间数据。我们把这些技术称为"整段代码生成"(whole-stage code generation)。

为了有个直观的感受，我们记录下在Spark 1.6和Spark 2.0中在一个核上处理一行的操作时间(单位是纳秒)，下面的表格能够体现出新的Tungsten engine的威力。

primitive	Spark 1.6	Spark 2.0
filter	15ns	1.1ns
sum w/o group	14ns	0.9ns
sum w/ group	79ns	10.7ns
hash join	115ns	4.0ns
sort (8-bit entropy)	620ns	5.3ns
sort (64-bit entropy)	620ns	40ns
sort-merge join	750ns	700ns

那么在新的Tungsten engine在端至端的查询表现又会咋样？我们比较了Spark 1.6和Spark 2.0在使用TPC-DS的基本分析，如下图：



除了whole-stage code generation可以提高性能，Catalyst方面也做了许多的工作，比如通用查询优化；还有一个新的矢量Parquet 解码器，它使得Parquet的扫描吞吐量提高了3x。

更加智能：Structured Streaming

Spark Streaming在大数据领域第一次尝试将批处理和流计算进行了统一。在Spark 0.7开始引入的第一个streaming
API称为DStream，它为开发者提供了几个强大的特性：仅一次的语义，大规模容错和高吞吐量。

然而，随着数百个真实的Spark Streaming部署后，我们发现，需要实时作出决策应用通常需要不止一个流引擎。他们需要深度地将批处理和流处理进行整合；需要和外部存储系统整合；以及需要应付业务逻辑变化的能力。其结果是，企业需要的不仅仅是一个流引擎;相反，他们需要一个完整的堆栈，使他们能够开发终端到终端的“持续的应用程序”。

一些人认为我们可以把所有的东西看作流。也就是说，提供一个编程模型，将批处理数据和流数据进行整合。

这个单一的模型有几个问题：首先，当数据到达时，对它进行操作将会变得非常难而且这会有许多限制性。其次，不同的数据分布，不断变化的业务逻辑和数据的延迟都增加了独特的挑战。第三、大多数现有系统中，例如MySQL或Amazon
S3中，不表现得像一个流；而且许多算法在流数据上无法工作。

Spark 2.0的Structured Streaming APIs是一种新颖的流处理方式。它的实现源于最简单地计算流数据的答案是不要想象它是一个流(这句话不太好翻译，自己看英文：the simplest way to compute answers on streams of data is to not having to reason about the fact that it is a stream

)。这个真理来源于我们的经验。结构化数据流的愿景是利用Catalyst优化器来发现什么时候可以透明的将静态的程序转到增量执行的动态工作或者无限数据流中。当我们从这个数据结构的角度来看到我们的数据，这就简化了流数据。

作为实现这一愿景的第一步，Spark 2.0附带了一个最初版本的Structured Streaming API（扩展自DataFrame/Dataset API），这个统一对现有的Spark用户比较容易适应，因为这让他们能够充分利用Spark batch API知识来解决实时中的问题。这里主要功能将包括支持基于event-time的处理、乱序/延时数据、sessionization以及非流数据sources和sinks的紧密集成。

Streaming显然是一个非常宽泛的话题，所以敬请关注databricks的博客对于Spark 2.0的Structured Streaming介绍，其中将会包括那些将会在此版本实现，哪些将会在未来版本实现。

总结

Spark用户最初转向Spark的原因是因为它的易用性和性能。Spark 2.0将付出双倍的努力来扩展它以使得它支持更广泛的workloads，我们希望你喜欢我们已经做的工作，并期待着您的反馈。

本文翻译自：https://databricks.com/blog/2016/05/11/spark-2-0-technical-preview-easier-faster-and-smarter.html