OneNet: High Performance Distributed Functional Computing Platform

Point of Contact: Jin Li (jinl@microsoft.com)

Abstract

OneNet is a high performance distributed functional computing platform. It is extensible for system builders (e.g., adding support to Azure is less than 250 lines of codes), natively supports computation across multiple clusters, and can mix and match specialized device compute (e.g., GPGPU/FPGA/Smart SSD) with CPU based cluster computing. OneNet can slash the engineering cost (including development, debugging and deployment) of building a complex distributed system to 1/3-1/5, significantly speed up a distributed system project, transforms large-scale data analysis, and enable a cost competitive cloud computing service. OneNet achieves these via distributed functional computing, so that 1) programmers can easily build a highly complex distributed system in functional computing form, and describing the problem as how dataset are transformed/related to each other, and 2) the program written can be native executed remotely, which enables extensively sharing of in-memory data within and across jobs, and lead to the capability of combining batch processing of big data and interactive real-time data analysis. Distributed functional computing will revolutionize distributed system building.


We are in the era of big data. The capability to quickly process huge amount of incoming stream of business data (click stream, transactions, sensor reading, business intelligence and social media interaction data) will give a critical competitive edge to the business. As the processing and I/O capabilities of a single machine have not kept up with the growth of the data, the only choice is to scale out the computation across a cluster.

However, distributed system is notoriously difficult to program, because of the difficulties during the programing stage: 1) to think about failures in distributed programming, 2) to consider the heterogeneity of a distributed cluster, 3) to reason about data locality in cluster programming, 4) to achieve consistency across cluster, and 5) to debug the developed program. The required knowledge, skillset and programming logics involved in distributed programming is highly sophisticated and complex, which makes the software engineers with distributed programming talent a highly prized target. To reduce the barrier to program a distributed cluster, a wide range of programming models have been introduced, such as MapReduce, Hadoop, DryadLINQ, SCOPE, Naiad and Orleans. There are also an array of point solutions for different distributed tasks, such as Mahoot and ScopeML for machine learning, GraphLab and Trinity for graph computation, etc.

Functional programming implements computation as nested evaluation of functions. It has long been considered a natural choice for big data analytics, as a data analytical/processing problem can be expressed as repeated transformations and actions onto the data. In fact, popular programming models prior have already used some/partial functional programming concepts. E.g., MapReduce/Hadoop program treats data analysis as a set of Map() and Reduce() expressions on the dataset. While DryadLINQ/Scope allows programmers to write lambda expressions as argument in LINQ/Scope queries (SQL-like, such as Select, GroupBy, etc..). Compared to prior programming models, a key distinction is that OneNet is a full distributed functional computing platform (build on top of F#, a functional programming language in .Net). The direct benefit is that OneNet allows the expression of
arbitrary functions/transforms/actions in distributed programming, rather than a restricted set of transforms (such as Map/Reduce in the Hadoop framework, and the query language and SQL-style in DryadLINQ/Scope framework). As a result, OneNet can build distributed systems that is not feasible due to the constraint of the prior programming models (Map/Reduce or query). For example, OneNet can be used for:

1) Multi-Cluster compute.
   By describing the computation as data transforms across clusters, OneNet can build a distributed computing platform that involves multiple clusters. This can be used to support hybrid public-private cloud compute, in which non-sensitive data can be computed in a public, shared cloud (e.g., Azure), while proprietary, business critical data can be computed in a private, secured cluster. The programmer can ensure that the proprietary data is never exposed to the public cloud.

2) Combine specific device (GPGPU/FPGA/Flash) compute with generic CPU compute
   Some of the cluster can contains special computation device, such as GPGPU, FPGA, Smart SSD that is unavailable in a generic cluster. Programmers simply writes the program as a transform module across the GPGPU/FPGA/Smart SSD device, or as a source function on GPGPU/FPGA/Smart SSD device. In OneNet, the special device doesn’t need to support all computing functionality. It only needs to perform those operations that can be efficiently executed on GPGPU/FPGA/Smart SSD, with the rest of the computation load picked up by CPU.

2. OneNet: High Performance Distributed Computing via Native Remote Execution

As a full distributed functional computing platform, OneNet is able to significantly reduce the engineering cost (including the development, debugging and deployment), and improves the performance of the distributed systems. Functional computing (such as F#) elevates function to a first class citizen in the programming language. Function variables can be assigned, taken as an argument to other functions, returned as result, serialized/deserialize (as closure) to remote machine for execution. Leveraging this, OneNet enables the execution of remote function natively in distributed programming. Consequently, when programming OneNet, the programmer has direct and fine-grain control over exactly how the program will be executed in the remote machines. As a result, OneNet can be used to build high performance, real-time analytical engines. Some of the unique capabilities of OneNet in this area are:

1) Extensive in-memory data sharing within & across jobs, hybrid batch processing and interactive real-time data analytics.

OneNet supports sharing in-memory data in multiple forms, both as byte stream in local memory or shared remote memory, or as native in-memory instantiated object. The second form of sharing instantiated object allow the job to access the data natively with high performance. Also, complex concurrent lock-free operations, such as compare and swap, add or update can be carried out on the shared data for very high performance operations. This form of data sharing can also enable big data analytical solution that combines both batch processing and real-time information. This is achievable because the batch processing job and the real-time analytical job can be programmed as separate jobs that share a common data store (e.g., of trained model, current state, etc..) The first form of byte stream sharing, on the other hand, provide better job isolation and preventing one job from crashing another. The disadvantage is that the shared data need to be deserialize and instantiated during the access, which has an impact on the performance.

2) Significant reduction of engineering cost (development, debugging, deployment)

By building a complex distributed system via nested evaluation of functions, OneNet naturally lead to a modularized distributed system, as each stage of evaluation becomes one individual module. When programming distributed
system via OneNet, it is comparable to building a complicated LEGO project from simple LEGO pieces. The modularity leads to the extreme extensibility. For example, to enable OneNet to support Azure, a key component is to enable the use of Azure Storage for storage stream. This requires less than 250 lines of code. We expect a similar amount of effort can allow OneNet to support other cluster storage, e.g., Cosmos Store, Hadoop/HDInsight, AutoPilot Drive, and/or FDS. In fact, byte stream sharing in local and/or remote shared memory is simply another implementation of the storage stream. Given OneNet’s capability to support multi-cluster compute, OneNet can be used to build a distributed computation platform that operates on the combined data of Azure Storage, Cosmos Store, AutoPilot Drive, FDS, etc..

Another aspect that make OneNet ideal for distributed system building comes from the static type and type inference feature of F#. Because F# is a strongly typed programming language, thereupon, a big class of type related bugs can be caught at compiler time. Through type interference, F# program can be written succinctly, which significantly reduces the line of code of the OneNet platform, and reduces the chances of errors in the development, especially as a complicated distributed system such as OneNet need to evolve and rewrite. The current OneNet code base is around ~40k line of code (about 2 man year). This is surprisingly small for such a generic distributed computing platform. I have engaged in other distributed system project in the past, e.g., P2P streaming (in C++, unmanaged code), erasure coded distributed storage (in C#). OneNet is much more complicated than any of those. I estimate that if OneNet is programmed in other language, such as C#, the engineering effort will increase by at least a factor of 3, if not more. Also, during the OneNet development process, the core execution engine has been rewritten 3 times. The typing interference and static typing greatly speed up the rewrite, and caught most of the bugs of incompatibility between code rewrite in compilation. When the rewritten program compiled through, the unit test that validates the OneNet operation runs through amazingly without a hitch.

3. **OneNet: Programmer’s (User’s) Perspective.**

At the top level, when writing an OneNet program, the programmer describes the dataflow of the program, via the use of data (either DSet or DKV), transforms and actions. This can be done best via F#, but can also be done in other programming languages (such as C#, python, etc.). The programmer will benefit from a basic understanding of functional programming concept, such as the closure, lambda expression and lazy execution. However, OneNet programming do not require the programmer to have extensive functional programming experience. In fact, most of the programming effort will be spent in writing modules, in which the programmer can write the program using his/her favorite language and/or tools. E.g., one of the OneNet unit test on image feature extraction uses Sho, IronPython, Intel MKL (Intel math kernel library, unmanaged code), and involves 17 DLLs, a dictionary of 200MB data, and environment variable setting. OneNet has the mechanism to properly distribute the DLLs, Data and environment variable setting to remote machine for execution. Also, during replication, OneNet checks the name and hash of the DLLs/Data, and if both name and hash are unchanged, the DLLs/data need not to be further replicated before execution. This greatly improves the efficiency of remote execution and deployment.

Five sample OneNet workflows that execute different big data analytical and processing tasks (distributed download and feature extraction, graph building and graph compute, SVM training, query serving) can be found in Figure 1. The compiled F# source code (not pseudo code!) that implements each workflow is no more than 50 line worth of F# code, which shows that OneNet allows a programmer to describe the complicated data flow of a distributed task in surprising succinct style. The programmer can reuse module that they have written previously in OneNet, in any managed code (via F#, C#, C++/CLI, IronPython, etc.) or even unmanaged code (C/C++). This enables the reuse of existing code base in OneNet based distributed programming as much as possible.
4. OneNet: Competitive Landscape

In the industry, the only other distributed functional computing platform is Spark (built by UC Berkeley AMP lab). Spark is widely touted as a unified distributed programming platform and execution engine that will push MapReduce/Hadoop to back burner. It is in use at Alibaba, Cloudera, IBM, Intel and Yahoo, etc. Databricks Inc, a startup with mission to transform large-scale data analysis by commercializing Spark, has raised $47 million (in 2 round of VC funding), and is considered by Ion Stoica (Professor at UC Berkeley) as “one of the most heated infrastructure deal I’ve ever been associated with”. Usually, when introducing Spark, the focus is on the in-memory data processing capability. However, I believe that it is the distributed functional programming model that is the corner stone of Spark, and allows many other components, e.g., Spark Streaming, BlinkDB, GraphX, MLBase to be quickly built upon Spark.

As a distributed functional programming platform, OneNet share many of the similar feature with the Spark. In fact, the dataflow programming model of OneNet looks similar to that of Spark. Compare with Spark, OneNet pushes the distributed functional programming aspects further than that of Spark, and enable a number of new cluster programming functionality, such as multi-cluster distributed programming, running of both managed code and unmanaged code, in-memory data sharing across jobs, push data flow, etc. OneNet also support more transforms and actions (e.g., IndexFold, which supports online query and index serve like operations). I believe that OneNet is more flexible and extensible than Spark, and will revolutionize how high performance distributed program is build in the future.

Figure 1. Five sample OneNet workflows. (F# code and workflow explanation can be found in the full white paper).