ZeroVM: Secure Distributed Processing for Big Data Analytics

Paul Rad[1, 3], Van Lindberg [3], Jeff Prevost [2], Weining Zhang [1], and Mo Jamshidi [2]
[1] Department of Computer Science, the University of Texas at San Antonio
[2] Department of Electrical Engineering, the University of Texas at San Antonio
[3] The Office of the CTO, Rackspace the Managed Cloud Company
{paul.rad, van.linberg}@rackspace.com, {jeff.prevost, weining.zhang}@utsa.edu, moj@wacong.org

***Abstract*—A key challenge for any large-scale computation today, whether in "big data" or in handling large-scale web services, has to do with the management of data. In the big data context, the arbitrary separation of storage and computation increases latency and decreases performance. ZeroVM is a lightweight container-based virtualization platform that provides deterministic process execution and isolation. The philosophy behind ZeroVM is to virtualize applications then move the application to the data. This provides the ability to transform or process data in situ, rather than moving data to where the application is located. With the ability to move and execute application next to data, ZeroVM changes the conventional wisdom on infrastructure centric commuting models and enables even more data centric computing models to be used for Big-Data Analytics. The ZeroVM distributed processing framework proposed in this paper presents new opportunities for processing, storing and using data, particularly in big data analytics.**

***Index Terms*— Big-Data; Cloud; Container; Data Analytics; Hadoop; Swift; Virtualization; ZeroVM**

I. Introduction

The rapid growth of structured and unstructured data-volumes presents a challenge to organizations tasked with management and storage of data. Efficient processing of huge structured and unstructured data sets in the cloud using commodity servers has become a new paradigm for computation [1, 2, 17]. This creates opportunities to process data either at acquisition time, such as with mobile devices, or at rest where it is stored offline. A key challenge for any large-scale computation today, whether in "big data" or in handling large-scale web services, has to do with the management of data [22]. In context of big data, the historic separation of storage and computation has the net effect to increase latency as well as decrease performance. In the services context, the management of backend data (such as in a database) introduces substantial complexity and scaling challenges.

Various technologies have been proposed and are being used to address these issues in different contexts. For example, Hadoop and HDFS are widely used for data processing because it addresses the issue of compute and data locality. Sharding, replication, and many "NoSQL" technologies have been developed to address many of the web service use cases. Nevertheless, the management of data, either as "big data" or in a shared state is a major pain point for widely distributed computation.

ZeroVM [6] is designed as a lightweight container-based virtualization platform. Code execution is through a Software Fault Isolation (SFI) sandbox based on the Google Native Client (NaCl) [5], which has gained substantial recognition for its secure process execution and isolation from the host operating system. The ZeroVM distributed processing framework proposed in this paper presents new opportunities for processing, storing, and using data, particularly in big data analytics. The philosophy behind ZeroVM is to virtualize applications and move them to the data, providing the ability to transform or process data in situ, where it is stored. As ZeroVM changes the conventional wisdom on infrastructure centric computing models by enabling the transition to data centric computing models, one of the most interesting use cases is the convergence of data and compute for Big-Data Analytics.

Although OpenStack Swift [13, 14] facilitates storage and management of almost an unlimited amount of data, the conventional cloud computing paradigm, for the purpose of processing, requires the enormous amount of data to be moved back and forth to the computing host. This exhausts significant CPU cycles and results in serious I/O bottlenecks. ZeroVM is a specially designed process isolation and management technology for the cloud and eliminates unnecessary movement of data by enabling data local computing. This is accomplished by virtue of uniquely designed applications called ZAP (ZeroVM Application Package) which facilitates computation near the data inside Swift storage itself.

OpenStack Swift is a highly deployed open source cloud storage solution. With its vast storage capability, it is used to store any number of large and small objects by way of its RESTful HTTP API. A user can submit a GET request to download a file and a PUT request to upload a file. A fundamental problem in the cloud storage system is that whenever data is required to be processed, it has to be moved to the computing hosts (ex. VM, EC2 instance) before computation and moved back to the original storage afterward which results significant I/O overhead.

ZeroVM [4], a specially built secure process manager for the cloud promises to solve the problem of secure computation. This is an application virtualization technique based on software fault isolation which is able to run

arbitrary (and potentially malicious) code and still provide a security guarantee.

Unlike existing solutions such as Docker [21], which is also very exciting in its own merit, ZeroVM focuses more on fault isolation, secure computation, and composition of many small processes to solve computing problems.

Apache Hadoop is another data processing/storage framework similar to ZeroVM. It is an open source implementation of Google's Map/Reduce [18, 10, 12] distributed computation framework on top of the Hadoop Distributed File System (HDFS). HDFS is a distributed file system, inspired by Google's Google File System (GFS) [19], that stores files, as blocks, across multiple DataNodes (data servers). A central NameNode manages the metadata associated with the files and which of the DataNodes contains the replicas of the data. Client applications (such as Hadoop) perform read operations by first interrogating the NameNode. The NameNode queries its internal database for the location of the request (i.e. determines which DataNodes contain the desired information) then the client creates a direct channel to the nearest DataNode containing the required blocks. Write operations are accomplished by the NameNode choosing the required number of DataNodes, based on the redundancy setting, and then handing this off to the client. The client then creates a channel to the first DataNode and the packets are streamed in a pipeline to the subsequent DataNodes [7]. Hadoop allows compute requests (mapping functions) to be placed on the DataNodes to create a parallel computing architecture, which speeds up access by leveraging concurrency. The benefit of the approach with ZeroVM presented in this paper is that the data can now be processed real-time concurrent, but sequential, as a very large stream of real-time data being stored in object storage (Swift). An example of the use of the proposed computing model is the real-time analysis of a large number of files, such as log files, or for pattern matching for any purpose in general (i.e. security).

This new technology, when integrated with Swift storage, is able to execute arbitrary applications (and potentially unsafe code) inside Swift clusters with the benefit of processing data locally. With its tight security guarantee, ZeroVM insures both the data owner and storage provider from the potential security risks arising from completely untrusted applications through enabling data local in-storage computation. This new paradigm introduces a broad set of opportunities. For example, it is now possible to search data while the data is in storage, look for patterns, or serve a file partially along with other exciting use cases like running query for big data and extracting salient customer patterns or product demands. The integration of these two new technologies also opens up an exciting era for both the cloud storage provider and the cloud customer. The storage provider can offer, in addition to storage, useful data processing applications which may help customers to get better service and even save money which was spent due to the movement of data. With ZeroVM, the customer no longer needs to provision large clusters for the processing of data.

The main contributions of this work are:

- a data-driven computing model derived on moving and executing user defined programs near the data inside of an object storage cloud
- a highly secure lightweight container-based platform through a software fault isolation (SFI) sandbox for packaging user defined programs
- a scheduler for dynamic provisioning of user defined programs inside object storage clouds

The remainder of the paper is organized as follows. Section II presents the ZeroVM architecture. Section III discusses ZeroVM's security and programing development model. Section IV concludes the paper with directions for future work.

II. Architecture

In this section we provide a short overview of the ZeroVM architecture in the context of secure, parallel processing of mass data stored in Swift. ZeroVM is a platform in which the following systems are integrated together:

• Swift, a highly available, distributed, eventually consistent object/blob store. Distributed as part of the OpenStack cloud computing system.

• ZeroVM, a Software Fault Isolation (SFI) based process sandbox with memory, computation, I/O monitoring, auditing, and control derived from the Google Native Client (NaCl) sandbox [3, 4]. Each one of these subsystems is described in further detail below.

A. OpenStack Object Storage (Swift)

Swift provides the underlying storage system and computational resources. The Swift architecture includes a proxy server, the "ring," (a distributed hash table lookup system), one or more storage policies, metadata servers (for account and container information), and object servers (for the objects themselves).

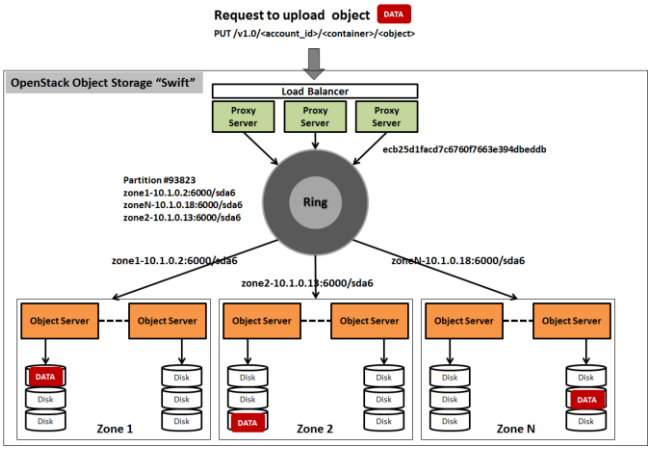

Figure 1: OpenStack object storage "Swift" architecture

Proxy Server: The Swift Proxy Server is responsible for

tying together the rest of the Swift architecture. For each request, it will look up the location of the account, container, or object in the ring (see below) and route the request accordingly. The public API is also exposed through the Proxy Server.

The Ring: A ring represents a mapping between the names of entities stored on disk and their physical location. There are separate rings for accounts, containers, and one object ring per storage policy. When other components need to perform any operation on an object, container, or account, they need to interact with the appropriate ring to determine its location in the cluster.

The Ring maintains this mapping using zones, devices, partitions, and replicas. Each partition in the ring is replicated, by default, 3 times across the cluster, and the locations for a partition are stored in the mapping maintained by the ring. The ring is also responsible for determining which devices are used for handoff in failure scenarios.

Object Servers: The Object Server is a blob storage server that can store, retrieve and delete objects stored on local devices. Objects are stored as binary files on the filesystem with metadata stored in the file's extended attributes. Various replication and auditing processes are designed to keep the system in a consistent state in the face of temporary error conditions like network outages or drive failures.

B. ZeroVM

ZeroVM consists of three conceptual parts: a secure process sandbox (the "cell") implemented using software fault isolation, a gateway daemon for creating external connections between a cell and other resources, and a compilation toolchain for creating applications loadable by the SFI cell [4] .

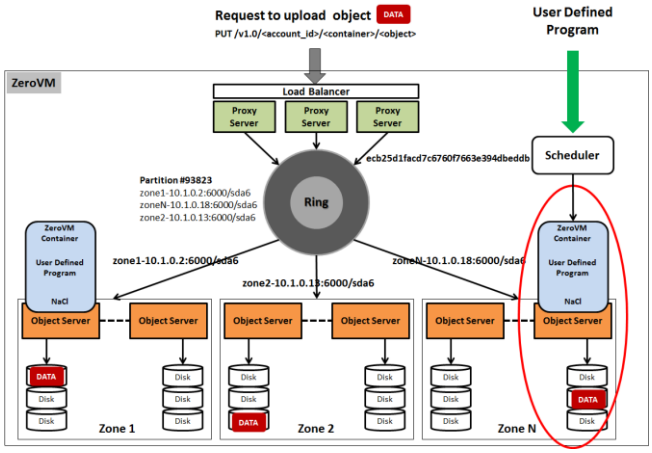

Figure 2: Swift architecture with addition of ZeroVM containers

ZeroVM cells are typically transient, set up in a virtual cluster for a single computation and then discarded. ZeroVM uses the following primitives: Manifests: A manifest is a statement of the capabilities that are to be provided to a particular cell. If a cell cannot be created that implements the required capabilities, the cell is never created. Manifests also allow particular resources (computational resources, memory, I/O) to be put within predefined limits. Channels: Channels are the key component of the ZeroVM I/O subsystem. On the host system side, channels are backed by the file system using regular files, pipes, character devices, or TCP sockets. On the guest side, each channel is a device file, which can either be used as a character or block device. Channels are the only way for untrusted code running within a ZeroVM cell to communicate with anything outside the cell. Access to and creation of channels must be declared ahead of time in the manifest. System Map: A configuration file passed to the scheduler describing a particular complex computation. A system map describes the connections that need to be made between different data sources and ZeroVM cells in order to accomplish the computing task.

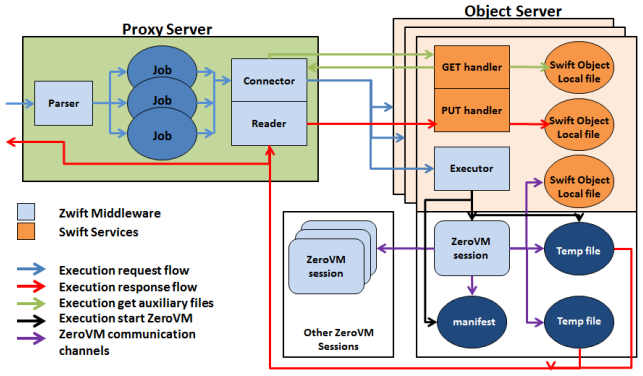

Figure 3: ZeroVM middleware architecture

ZeroVM is a middleware package for OpenStack Swift. This provides the capability to run containerized applications directly on object storage servers. It includes the system scheduler and can be used to initiate computations as described in a ZeroVM system map.

III. Security and Programing Model

ZeroVM's security is conceptually based on Software Fault Isolation (SFI), while the implementation is derived from the Chromium Native Client (NaCl) project sponsored by Google for use in the Chrome web browser. Software Fault Isolation is based upon the analysis of program machine code in ways that maintain the semantic correctness of the code but otherwise modify the programs so that they behave only in safe ways.

A. Secure Process Virtualization

ZeroVM uses software fault isolation (SFI) to enable the execution of untrusted native code inside a ZeroVM container, giving applications greater access to the performance of the object storage node while avoiding the security problems with current infrastructure for plugins. However, Native Client is not appropriate for modules requiring process creation or unrestricted access to the network. Trusted facilities such as storage should generally be implemented outside of Native Client, encouraging simplicity and robustness of the individual components and enforcing stricter isolation and scrutiny of all components. This design choice echoes microkernel operating system design [16].

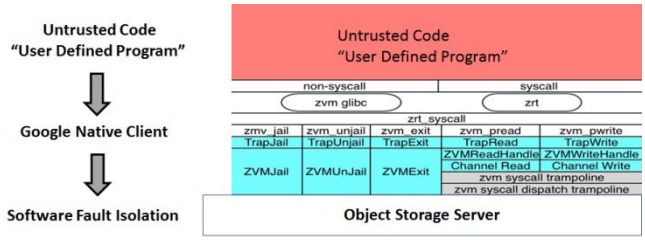

Figure 4: ZeroVM security architecture based on NaCl

While acknowledging the insecurity of the current systems for incorporating native-code into containers, we also observe that there is no fundamental reason why native code should be unsafe. In Native Client, we separate the problem of safe native execution from that of extending trust, allowing each to be managed independently. Conceptually, Native Client is organized in two parts: a constrained execution environment for native code to prevent unintended side effects, and a runtime for hosting these native code extensions through which allowable side effects may occur safely [15, 16] .

In most security implementations, the environment is responsible for policy enforcement, with the enforcement mechanism being provided through a language runtime, an OS, a kernel, or the hardware. Hardware methods include addressing mechanisms; software methods include having multiple modes where different modes have different levels of access [3] . Software Fault Isolation differs from environmental security approaches by constructing a piece of software that transforms a given program p into a program p', where p' is guaranteed to satisfy a security policy of interest. ZeroVM includes several parts, including a compiler toolchain, a binary verifier, and a loader that takes a compiled and verified binary and executes it within an SFI "cell." The cell is further modified to disallow any direct access to system calls and instead to redirect those system calls to a security proxy. Only a limited number of calls (currently five) are proxied through to the underlying system, and all other system calls are implemented as part of a lightweight runtime running in untrusted mode within the ZeroVM cell.

Unlike some other sandboxing systems, ZeroVM does not require the use of a specific language. Cells also allow low-level code, including C and assembly. This foundation allows both highly performant and highly convenient code choices depending upon the need.

B. Application Development for ZeroVM

The Native Client (NaCl) system allows developers to write components of their applications in C++ or other languages that compile to object code, to access secure APIs to system services and to execute these components on Swift storage servers without sacrificing the security properties users expect from the object storage. ZeroVM creates a container around a single process, using technology based on NaCl. The programs executed in ZeroVM container must first be cross-compiled to the NaCl platform.

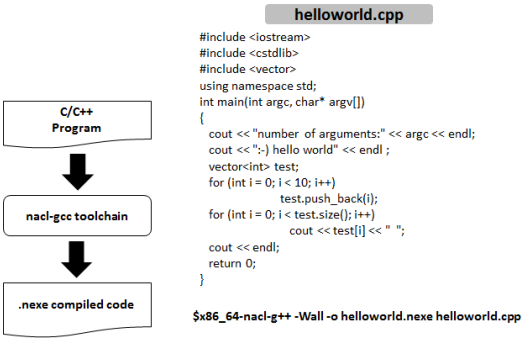

Figure 5: Application development phase using ZeroVM toolchain

Figure 5 shows the basic workflow in compiling an executable using *nacl-gcc* toolchain. NaCl modules can thus harness the full power of the object storage servers' CPUs and other resources, but to do so, a compiled NaCl module must necessarily be tied to a specific Instruction Set Architecture (ISA). While the operating system neutrality of NaCl tends to encourage good practices with respect to ISA portability, the burden of building, testing and deploying a program on all supported hardware platforms lies with the developer.

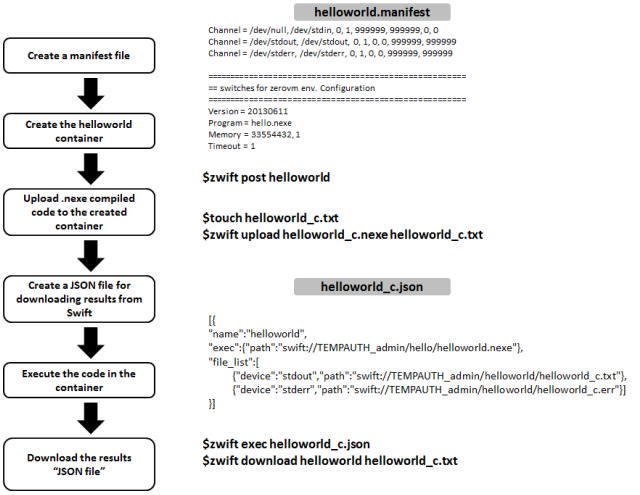

Figure 6: Program execution process

The second stage of the process as shown in figure 6 moves a ZeroVM container, consisting of an application compiled into a NaCl executable, into a Swift object storage server for execution.

IV. FUTURE WORK

This section presents some of the future work that the ZeroVM team at UTSA is considering; ZeroVM being an open source project implies that new features and changes are decided by the development community at large.

A. Dynamic Scheduling with Resource Discovery

Another aspect of both the security and usability is that the system is designed to take advantage of the existing computational resources available, while responding flexibly to changes in the underlying infrastructure. As a consequence, ZeroVM processes are designed to be largely

ephemeral, with individual processes being instantiated, evaluated, and then discarded in response to each request. The overhead for each secure process is minimal, with each SFI cell starting in as little as 5 milliseconds. To model complex applications, multiple secure processes are composed into a computational graph that is evaluated in real time relative to the discovered available computational and storage resources. Code is loaded into the graph and executed within the ZeroVM cells based upon a dynamic scheduling algorithm that takes into account data locality and available resources [8, 9, 11]. The programming environment exposed within the cell, however, does not change regardless of the location where it is executing.

B. Portable JIT Services for ZeroVM

As ZeroVM provides isolates processes using Software Fault Isolation (SFI) based on Google's Native Client [2]. Software Fault Isolation works by taking machine code and ensuring that every potentially dangerous operation in the code is guarded by dynamic checks that ensure that errors do not occur that violate the systems invariants. ZeroVM also plans to support loading portable bitcode programs [5]. What has not yet been done is for either Google Chrome or ZeroVM is supporting the dynamic loading of PNaCl code [3]. This would allow tenants to run performant JIT-based implementations of languages like Java in ZeroVM containers without needing to be concerned with the actual architecture of the servers hosting the ZeroVM containers.

C. Cloud Media Computing for Large Scale Image Search

In this project, we focus on Cloud Media Computing for Large Scale Image Search. Currently, Cloud Computing has become a scalable service consumption and delivery platform. Meanwhile, the explosion of multimedia data brings us new challenges with regards to image/video content analysis and image search. Multimedia content distribution, searching and sharing across heterogeneous devices, such as mobile phones, pose a great motivation to adapt conventional search strategy to cloud media computing platform. We will explore how multimedia data and search strategy will be optimized leveraging ZeroVM architecture, and how media content will be distributed and searched ubiquitously across heterogeneous devices.

V. Acknowledgment

The authors would like to thank all members of the ZeroVM team at Rackspace for their hard work building ZeroVM. We would like to thank all ZeroVM code committers and specially Ron Pedde, William Kelly, Ryan Mckinney, and Constantine Peresypkin from Rackspace for technology and trend discussions and valuable comments.

VI. References

[1] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," Communications of the ACM, vol. 51, no. 1, pp. 1–13, 2008.

[2] T. White, Hadoop: The Definitive Guide. O'Reilly Media, Inc., 2012.

[3] J. Ansel, P. Marchenko, U´ . Erlingsson, E. Taylor, B. Chen, D. Schuff, D. Sehr, C. Biffle, B. Yee. Language-Independent Sandboxing of Just-in-Time Compilation and Self-Modifying Code. ACM SIGPLAN Conference on Programming Languages Design and Implementation, p. 355–366, 2011.

[4] D. Sehr, R. Muth, C. Biffle, V. Khimenko, E. Pasko, K. Schimpf, B. Yee, B. Chen. Adapting Software Fault Isolation to Contemporary CPU Architectures. Proceedings of the 19th USENIX Conference on Security, p. 1–12, 2010.

[5] Stability of the PNaCl bitcode ABI. Document version 0.1. June 17, 2013.https://sites.google.com/a/chromium.org/dev/nativeclient/pnacl/stability-of-the-pnacl-bitcode-abi

[6] ZeroVM. http://zerovm.org/wiki/The_Cloud_Hypervisor

[7] K. Shvachko, H. Kuang, S. Radia, R. Chansler, "The Hadoop Distributed File System", in Proceedings of IEEE Conference on Mass Storage Systems and Technologies (MSST), 2010.

[8] S. Govindan, J. Liu, A. Kansal, and A. Sivasubramaniam, "Cuanta: quantifying effects of shared on-chip resource interference for consolidated virtual machines," in Proceedings of the 2nd ACM Symposium on Cloud Computing. ACM, 2011, p. 22.

[9] M. Zaharia, D. Borthakur, J. Sen Sarma, K. Elmeleegy, S. Shenker, and I. Stoica, "Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling," in Proceedings of the 5th European conference on Computer systems. ACM, 2010, pp. 265–278.

[10] M. Zaharia, A. Konwinski, A. D. Joseph, R. H. Katz, and I. Stoica, "Improving mapreduce performance in heterogeneous environments." in OSDI, vol. 8, no. 4, 2008, p. 7.

[11] B. Palanisamy, A. Singh, L. Liu, and B. Jain, "Purlieus: locality-aware resource allocation for mapreduce in a cloud," in Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis. ACM, 2011, p. 58.

[12] M. Zaharia, D. Borthakur, J. S. Sarma, K. Elmeleegy, S. Shenker, and I. Stoica, "Job scheduling for multi-user mapreduce clusters," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2009-55, 2009.

[13] OpenStack Swift. http://www.openstack.org/

[14] Milojičić, Dejan, Llorente, Ignacio M.; Montero, Ruben S.: OpenNebula: A Cloud management Tool. Journal IEEE Internet Computing. 2011. vol 15. issue 2. p. 11-14

[15] NaCl project: Disabling sources of non-determinism for guest code. http://code.google.com/p/nativeclient/wiki/DeterministicExecution.

[16] B. Yee, D. Sehr, G. Dardyk, J. B. Chen, R. Muth, T. Ormandy, S. Okasaka, N. Narula, and N. Fullagar. Native Client: A Sandbox for Portable, Untrusted x86 Native Code. In 30th IEEE Symposium on Security and Privacy, pages 79-93, 2009.

[17] Sage A. Weil , Andrew W. Leung , Scott A. Brandt , Carlos Maltzahn, RADOS: a scalable, reliable storage service for petabyte-scale storage clusters, Proceedings of the 2nd international workshop on Petascale data storage: held in conjunction with Supercomputing '07, November 11-11, 2007, Reno, Nevada

[18] MapReduce: Simplified Data Processing on Large Clusters. Available at http://labs.google.com/papers/mapreduceosdi04.pdf
[19] The Google File System. Available at http://labs.google.com/papers/gfs-sosp2003.pdf
[21] Docker. www.docker .com
[22] M. Jamshidi and B. Tannahill, “Big Data Analytic: From PCA to Deep Learning” Proc. AAAI Workshop, Stanford University, Stanford, CA, March 25, 2014.