International Journal of Fog Computing
Volume 2 • Issue 1 • January-June 2019

Overview of Big Data-Intensive Storage and its Technologies for Cloud and Fog Computing

Richard S. Segall, Arkansas State University, Jonesboro, USA

Jeffrey S Cook, Independent Researcher, Paragould, USA

Gao Niu, Bryant University, Smithfield, USA

ABSTRACT

Computing systems are becoming increasingly data-intensive because of the explosion of data and the needs for processing the data, and subsequently storage management is critical to application performance in such data-intensive computing systems. However, if existing resource management frameworks in these systems lack the support for storage management, this would cause unpredictable performance degradation when applications are under input/output (I/O) contention. Storage management of data-intensive systems is a challenge. Big Data plays a most major role in storage systems for data-intensive computing. This article deals with these difficulties along with discussion of High Performance Computing (HPC) systems, background for storage systems for data-intensive applications, storage patterns and storage mechanisms for Big Data, the Top 10 Cloud Storage Systems for data-intensive computing in today's world, and the interface between Big Data Intensive Storage and Cloud/Fog Computing. Big Data storage and its server statistics and usage distributions for the Top 500 Supercomputers in the world are also presented graphically and discussed as data-intensive storage components that can be interfaced with Fog-to-cloud interactions and enabling protocols.

KEYWORDS

Cloud Storage, Data-Intensive, Fog-To-Cloud Computing, High Performance Computing (HPC), Key-Valued, Message Passing Interface (MPI), Storage Systems, Supercomputers

INTRODUCTION

Data-intensive computing systems have penetrated every aspect of people's lives. Behind it is the scientific and commercial processing of massive data impacting the decision makings in companies, academics, governments, social cites, and personal lives.

There are two types of data-intensive computing systems that continue to co-exist in the modern computing environment:

1. High Performance Computing (HPC) systems, consisting of tightly coupled computer nodes and storage nodes that are used to execute task parallelism for scientific purposes like weather forecasting, physics simulation, and the likes. (Rouse, 2017b).

DOI: 10.4018/IJFC.2019010104

2. Message Passing Interface (MPI) is an example of a computing framework on HPC systems. Big Data systems, comprised of more loosely coupled nodes, are used to execute data parallelism for tasks such as sorting, data mining, machine learning, etc. MapReduce is an example of a computing framework on Big Data systems. ((Barney, 2017) (Rouse, M. (2017c)).

Both HPC systems and Big Data systems that are deployed for multiple users and applications to share the computing resources so that 1) the resource utilization is high, driving down the usage cost per application/user, and the users get better responsiveness of application execution; 2) the data set is reused without extra overhead to move around performing redundant Input/Outputs (I/O) and users can also save space.

As the computing needs continue to grow in data-intensive computing systems, the shared usage model results in a highly resourceful competing environment. For example, Amazon, Apple and eBay provides HPC and Big Data as cloud services. Hadoop version 2, YARN (Yet Another Resource Negotiator), that is one of the key features in the second-generation Hadoop 2 version of the Apache Software Foundation's open source distributed processing framework. Originally described by Apache as a redesigned resource manager. YARN is now characterized as a large-scale, distributed operating system for Big Data applications which provides a scheduler to incorporate both MapReduce and MPI jobs. (Rouse, 2017a).

As the number of concurrent data-intensive applications and the amount of data increase, application I/O's start to saturate the storage and interfere with each other, and storage systems become the bottleneck to application performance. Both HPC and Big Data systems I/O amplification adds to the I/O contention in the storage systems. To counter failures in these distributed systems, HPC systems employ defensive I/O's such as check pointing to restart an application from where it fails, and Big Data systems replicate persistent data by a factor of k, which grows with the scale of the storage system. Both mechanisms aggravate the I/O contention on the storage. The storage systems can be scaled-out, but the compute to storage node ratio is still high, rendering the storage subsystem a highly contended component (Xu, 2016). Therefore, the lack of I/O performance isolation in the data-intensive computing systems causes severe storage interference which compromises the performance target set by other resource managers proposed or implemented in a large body of works. Failure to provide applications with guaranteed performance has consequences. Data-intensive applications must complete in bounded time so as to get meaningful results. For example, weather forecast data is much less useful when the forecasted time has passed. Paid user in a Big Data system also require a predictable runtime even though the job is not time sensitive, and the provider may get penalized in revenues if jobs fail to complete in a timely manner. (Xu, 2016).

This chapter addresses the problems stated above for data-intensive computing systems. It provides different approaches for both HPC storage systems and Big Data storage systems because their differences in principles, architecture, and usage pose distinct challenges. Before studying these systems and addressing their respective problems separately, the discussion of the differences between these two types of systems is established here. (Xu, 2016).

HPC systems are strongly coupled distributed systems, connected by expensive hardware and network links (e.g. InfiniBand [inf]). The application execution principle focuses on *task parallelism*, and thus both its parallel compute processes and I/O requests are tightly coupled and must be executed *together*. This means a failure of any node results in the failure of the entire application. This is also why the check pointing I/O's are major sources of I/O's when running such applications, as the periodical save of application progress constitutes much higher amount of data than its original input and final output. (Xu, 2016).

The most widely used programming framework for HPC systems is Message Passing Interface (MPI) that is also discussed in this article.