DOI

https://doi.org/10.1109/BigData.Congress.2014.93

CloudJet4BigData: Streamlining Big Data via an accelerated socket interface

Frank Zhigang Wang
School of Computing
University of Kent
UK
Contact author:
frankwang@ieee.org

Theo Dimitrakos
British Telecommunication
UK

Na Helian
SCS
University of Hertfordshire
UK

Sining Wu
Xyratex
Havant
UK

Ling Li
SoC
University of Kent
UK

Rodric Yates
Hursley Lab.
IBM
UK

Abstract— Big data needs to feed users with fresh processing results and cloud platforms can be used to speed up big data applications. This paper describes a new data communication protocol (CloudJet) for long distance and large volume big data accessing operations to alleviate the large latencies encountered in sharing big data resources in the clouds. It encapsulates a dynamic multi-stream/multi-path engine at the socket level, which conforms to Portable Operating System Interface (POSIX) and thereby can accelerate any POSIX-compatible applications across IP based networks. It was demonstrated that CloudJet accelerates typical big data applications such as very large database (VLDB), data mining, media streaming and office applications by up to tenfold in real-world tests.

Keywords-Big Data, Cloud Computing, Cloud Storage, Protocols for networked storage systems, Portable Operating System Interface (POSIX)

I. INTRODUCTION

Big data needs to feed users with fresh processing results. Cloud Computing continues to become more data-intensive and cloud platforms can be used to speed up big data applications. Researchers from Berkeley listed top 10 obstacles and opportunities for Cloud Computing, of which at least four obstacles are storage-related [1]. Obstacle No.4 in their list is Data Transfer Bottlenecks that is also our research focus in this paper.

As an important domain of Cloud Computing, Cloud Storage is Internet-based storage involving virtual servers that are generally hosted by third parties. Storing data in this way offers near unlimited storage and can provide significant cost savings as there is no need for the business to buy, run, upgrade or maintain data storage systems with unused spare capacity. Internet is the communication channel for Cloud Storage. However, attempting to share or access data across the Internet/clouds often proves to be a frustrating and time-consuming experience. The "chatty" nature is negatively affected by WAN latency (normally ≥2 ms): after the packets are out on the fiber, the transmitter must stop until it gets an acknowledgement for those packets; after the round-trip delay time after starting, the acknowledgement gets back to the sender and the second burst with the default buffer size can be transmitted. As a result, applications using a single socket stream will not fill the communication pipe, i.e., the traffic load is less than its capacity. When the buffer size is taken as its default value of 64 kB (Linux/Unix), the capacity efficiency is about 1% [2]. Computer networks/grids/clouds are often characterized with a network latency of 1-2 ms (Local Area Network, LAN), 2-60 ms (Metropolitan Area Network, MAN) and 60-1000 ms (Wide Area Network, WAN), respectively [2].

The gap is widening between the maximum-achievable network bandwidth and the actually-utilized bandwidth [3][4]. The network bandwidth is doubling every 9 months, which even outperforms Moore's law [5]. As reported in TimesOnline, the Internet could soon be made obsolete [6]. The scientists who pioneered it have now built a fast replacement at speeds about 10,000 times faster than a typical broadband connection. That network, in effect a parallel internet, runs from CERN to 11 centers in Europe, United States, Canada, the Far East [7], and around the world. By contrast, the grids/clouds have been built with dedicated fiber optic cables and modern routing centers, meaning there are no outdated components to slow the deluge of data [6].

The primary cause of the gap is the conservative design of today's Internet infrastructure. The internet has evolved by linking together a large number of cables and routing equipment, much of which was originally designed for telephone calls and therefore lacks the capacity for high-speed data transmission. In general, the design of today's network infrastructure has been influenced by the need to enforce fair sharing of precious network resources. For this reason, today's Internet behavior is unnecessarily conservative for data-intensive applications such as Cloud Computing/Big Data on high bandwidth networks.

Most of today's network routing protocols select only a single path between a source and a destination. The multipath routing takes advantage of path diversity and achieves an aggregate bandwidth that is theoretically enormous [7]. This also means there is a huge bandwidth gap to fill.

Under some circumstances, distributing storage is more sensible than centralizing it. Looking at the present trends in network and storage technology, McGarrah observes that network bandwidth has grown at a much slower pace than disk storage capacity [8]. How to increase the efficiency of the network bandwidth over the storage capacity is a great concern.

There were a lot of efforts in addressing remote storage. The Internet Backplane Protocol (IBP) provides a C API and a tool set to automate the finding and usage of remote storage [9]. In GRID codes [10], strip-based erasure codes with high fault tolerance was developed for storage systems. With today's congestion control policy, parallel Sockets/TCP flows steal bandwidth from competing flows. A fractional parallel streams was reported to solve this fairness problem [11]. In [12], it is proposed to reorder aggressive block for large file transfers via FTP (File Transfer Protocol). Our developed

GOS Filesystem (GOS-FS) integrates a par-allel stream engine and Grid Security Infrastructure (GSI) [13][14].

This work investigates how a storage-networking protocol can best utilize the increased transfer bandwidth. Cloud Storage/Big Data appliances need to use a specific storage networking protocol to provide access to its resident data with end stations on the computational clouds. Our CloudJet protocol is designed for such a purpose.

II.CloudJet: A protocol for Big Data in clouds

As a variant or successor of NAS (Network-attached Storage), Cloud Storage appliances need to use a specific storage networking protocol to provide access to its resident data with end stations in the clouds. A CloudJet interface is designed to counterattack the network latencies in a typical big data environment.

2.1 CloudJet Objectives

Driven by the above identified problem of network latency in the clouds, a Socket-level interface titled “CloudJet” is specially designed for long-distance, large volume big data applications in the clouds through the efficient usage of the existing data communication protocols. CloudJet maintains multiple bidirectional process-to-process communication flows across an IP based network, such as the computational clouds. Our original contributions in this paper are: 1. A dynamic multi-stream/multi-path engine; 2. An encapsulating at the Socket level.

In 2001, Fisk et al predicted “Perhaps, parallel DRS (Dynamic Right-Sizing) streams will combine the best of both approaches (a single stream with dynamic buffer size and multi streams)” [15]. In this work, we have implemented this engine, in the form of a Socket protocol, preliminarily and are also extending it to multi-path scenarios. This Socket protocol is superior to what is used in (multi-streamed) GridFTP or Bit-Torrent. CloudJet encapsulates the above dynamic multi-stream/multi-path engine, which conforms to POSIX and thereby can accelerate any POSIX-compatible applications across IP based networks. In contrast, (multi-streamed) GridFTP or Bit-Torrent cannot be used to accelerate other applications as they themselves are applications.

In this CloudJet interface, a third layer of SocketMultiplier is developed and inserted between traditional BSD Socket and INET Socket (Fig.1) to form a 3-in-1 CloudJet (Fig.1). As the name implies, this SocketMultiplier opens multiple INET Sockets and enables simultaneous communication of multiple TCP streams or UDP datagrams on the same network channel. Data dividing/assembling has been integrated in SocketMultiplier. With the accelerated CloudJet that remains fully compatible with the existing IT infrastructures (BSD socket within POSIX), computer users can share remote big data in the clouds without having to adjust the default configurations or change the way they work (Fig.2).

Applications access the Network from BSD socket interface layer that is close to user space. INET Socket is an obvious entry points where Kernel handles various system calls for user space socket operations. There are three INET socket types: 1. Datagram sockets (connectionless, which use UDP); 2. Stream sockets (connection-oriented, which use TCP or SCTP); 3. Raw sockets.

In particular, we are interested in a dynamic model for end-to-end available bandwidth estimation, which helps us to design a new 3-in-1 socket with an automatic optimizer to dynamically adjust the number of sockets/streams or sockets/datagrams to take full advantage of the available bandwidth and also track the network bandwidth fluctuation in the clouds.

From its inception, the proposed CloudJet is designed to deal with long-distance, large volume, cross-domain big data operations in the clouds – a capability lacking in the current versions of the Socket interface that normally opens one INET socket and thereby enables communication of one TCP stream or UDP datagram. Applications over CloudJet will alleviate network latency issues for most big data applications.

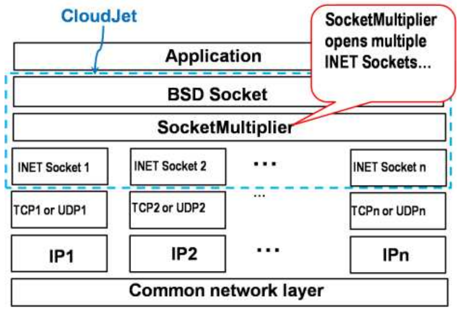
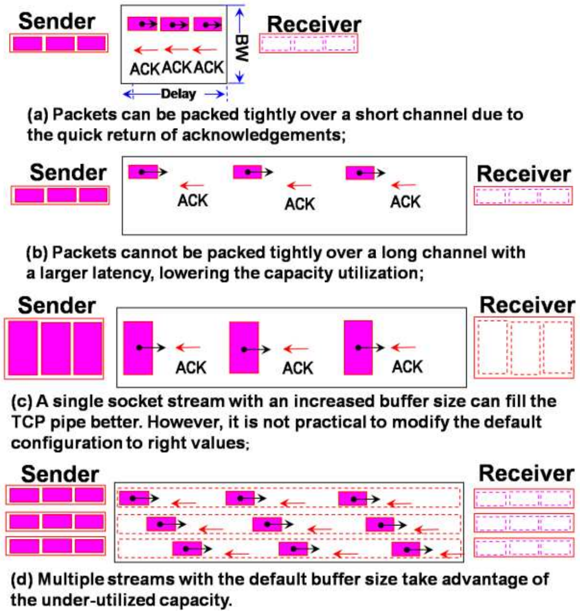


Fig.1 SocketMultiplier will be inserted between the traditional BSD Socket and INET Socket to form a 3-in-1 CloudJet.



(a) Packets can be packed tightly over a short channel due to the quick return of acknowledgements;

(b) Packets cannot be packed tightly over a long channel with a larger latency, lowering the capacity utilization;

(c) A single socket stream with an increased buffer size can fill the TCP pipe better. However, it is not practical to modify the default configuration to right values;

(d) Multiple streams with the default buffer size take advantage of the under-utilized capacity.

Fig.2 Single or multiple TCP socket streams with different capacity utilizations.

2.2 CloudJet Design & Implementation

As mentioned earlier, the original TCP protocol is illsuited to high-bandwidth, high-RTT networks and the actual network bandwidth utilization is unsatisfactory. In principle, we could list at least three typical techniques (Fig.2) to fill the empty pipes, which are (1) Increasing the buffer size; (2) Utilizing multiple streams with the default buffer size; and (3) Modifying the current congestion control algorithm. Fig.2 is just an example for TCP communications and the principle illustrated is also applicable to UDP communications. As well known, UDP is an unreliable protocol and should produce less overhead than TCP, therefore, the sent UDP throughput should be greater than the TCP throughput.

Option 1: Increasing the buffer size

In Fig.2, the shaded areas represent packet size and the large empty rectangles represent TCP pipe capacity (bandwidth*delay). As shown in Fig.2(a), a short pipe can pack packets tightly due to the quick return of acknowledgements. That is why even a single connection can fill the pipe at RTT=0. In Fig.2(b), a long pipe (with a larger latency) cannot pack packets tightly, i.e. network latencies result in the underutilization of network bandwidth resources. Increasing the effective size of the receive window is necessary to achieve high throughput for connections across the networks. As shown in Fig.2(c), a single socket stream with an increased buffer size can take advantage of the under-utilized capacity. Nowadays, both Windows (since Vista) and Linux (since 2.4) have support for TCP receive window scaling and autotuning.

Unfortunately, it is not practical to modify the default size of buffers to a right value. We had better use default configurations without applying any optimization. This is the usage condition for most scientists that normally have neither the needed expertise nor the time to configure the tools with high levels of optimization.

In order to avoid a tedious process of manually tuning system buffers, it is possible to set the size automatically at connection set-up. However, the buffer sizes are only appropriate at the beginning of the connection's lifetime. To address this problem, an automated dynamic right-sizing (DRS) technique throughout the connection's lifetime was proposed [15].

Option 2: Utilizing multiple streams with the default buffer size

Multiple streams with the default buffer size (Fig.2(d)) can also take advantage of the under-utilized capacity although each individual connection does not pack packets tightly. Examples include GridFTP [16] and SCTP (Stream Control Transmission Protocol) [17]. Our test result supports the claim that the multi-streamed data transport is an effective way to fill the TCP pipe and improve the capacity utilizations [13]. Most importantly, each individual connection in parallel streams still sticks to classic acknowledgement mechanism.

Option 3: Modifying congestion control algorithm

To overcome the slow-recovery in TCP Saw-Tooth behavior, TCP Vegas or Fast TCP uses queuing delay instead of loss probability as a congestion signal [18]. A Fast TCP flow seeks to maintain a constant number of packets in queues throughout the network. A single flow Fast TCP achieved an average throughput of 925 Mb/s over a 1Gbps link and utilization of 95% [18]. Scalable TCP [19] has been designed to ensure resource sharing and stability while maintaining agility to prevailing network conditions. A working multipath TCP congestion control algorithm [20] can seamlessly balance traffic over 3G and WiFi radio links. Nevertheless, either Fast TCP, Scalable TCP or Multipath TCP with an optimized congestion control is a solution at the new network transport layer, which is below our socket layer solution to be proposed. That is to say, the solutions at these two different layers may work together to aggregate the acceleration (Fig.1).

Our Option: A Socket protocol encapsulating a dynamic multi-stream/multi-path engine

CloudJet opens multiple INET sockets and uses parallel TCP streams or UDP datagrams to achieve very high transfer rates at a fraction of the memory cost and an overhead of managing multiple threads for data dividing/assembling. The application data are partitioned into segments with a fixed size. The segments are allocated to different streams or datagrams using Round Robin (or similar). Each thread is in charge of one or more streams or datagrams. The segments are transmitted simultaneously and then reassembled by the receiver. The final data is presented to applications as if they were transmitted through a single socket. Applications using CloudJet is believed to obtain near optimal TCP/UDP performance without having to adjust the default configuration or change the way they work.

In addition, there are another two advantages with our solution: 1. Fast Recovery; 2. CloudJet implementation at the socket level efficiently uses the existing data communication protocols such as Multipath TCP. The details will be discussed in Section 3.4 (Fast-Recovery Effect of Multi Streams) and Section 3.5 (Multi-path Routing increases the aggregate bandwidth), respectively.

2.3 Multi-stream Bandwidth Model and Dynamic Optimizer

Fisk et al found that, under some circumstances having a single stream with appropriately sized buffers, a similar performance may be achieved as parallel streams [15]. It is suggested that parallel DRS streams will combine the best of both approaches [15]. Bullet proposed an algorithm that sends data to different points in the overlay targeting high-bandwidth data distribution for applications include large-file transfers and real-time multimedia streaming [21]. In 2009, Kosar et al proposed a selection algorithm to balance buffer size and parallel stream number [4].They found that tuning buffer size and using parallel streams allow improvement of TCP throughput at the application level. Their preliminary results based on a ns-2 simulation show that using parallel streams on tuned buffers result in significant increase in throughput [4]. That is to say, tuning buffer size first and then the number of streams is better than tuning the number of streams first and then buffer size. In order to facilitate the above network setting tunings, it is possible to set up a service that makes the task of network tuning trivial for application developers and users [3].

In this work, we have implemented CloudJet and this data engine is superior to the above application-level work of tuning (multi-streamed) in terms of universally accelerating all POSIX-compatible applications.

III. Innovations & Comparison With Prior Work

3.1 Socket interface supports a much larger number of cloud applications

Note that (Socket-based) CloudJet differs from (File-based) GOS. Although both the VFS interface and the Socket interface are parts of POSIX, the Socket interface supports a much larger number of storage-networking applications than the VFS one in a typical cloud environment.

3.2 Socket level implementation in comparison with transport level one

In contrast to SCTP [17], Fast TCP [18] and Scalable TCP [19] and Multipath TCP [20], CloudJet implementation at the socket level is believed to have following advantages: (a) CloudJet is a Socket-level interface, rather than a new protocol, that efficiently uses the existing data communication protocols; (b) CloudJet remains fully compatible with the existing BSD socket, legacy applications can be simply deployed without having to adjust the default configurations or change the way they work; (c) Different from SCTP that provides an extended interface to use multi-streaming, multi-socketing/streaming in CloudJet is transparent to end applications as if the data was transmitted through a single socket; (d) CloudJet enabling multi-TCP-streams is still loss-based, avoiding the complex interactions between loss-based (traditional TCP) and delay-based protocols (like Fast TCP) when they share the network. (e) Socket level implementation facilitates application-specific optimization in contrast to the transport level implementation.

3.3 Dynamic Numbering Optimizer on a Lossy Network

As introduced in Section 2.3, Fisk et al suggested that parallel DRS (Dynamic Right-Sizing) streams will combine the best of both single stream with DRS and multiple streams [15]. In 2009, Kosar et al proposed a selection algorithm to balance buffer size and parallel stream number [4].

In a typical cloud environment, the packet loss rate p is a primary factor in determining aggregate TCP throughput of a parallel TCP connection session. Experience has shown that n parallel streams can dramatically improve application throughput, but random packet losses (<0.001) usually occur in one stream at a time [13][14]. Packet loss may be due to random factors other than network congestion, such as intermittent hardware faults. In theory, their effect on the aggregate throughput will be reduced by a factor of n due to the reduced overhead of re-sending the lost packets. When competing with connections over a congested link, each of the parallel streams will be less likely to be selected for having their packet dropped, and therefore the aggregate amount of potential bandwidth is reduced. The behavior of packet loss and its effect on the Multi-stream Bandwidth Model should be studied.

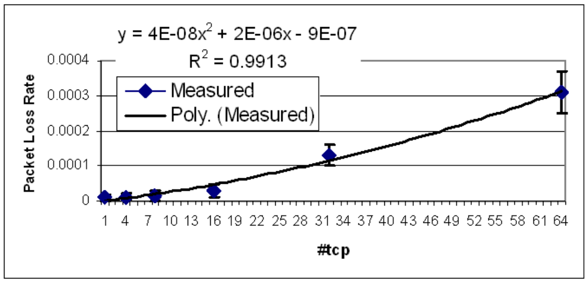


Fig.3 The statistical differences between the measured and estimated packet loss rates. An empirical formula with a confidence coefficient R is obtained in the form of a complete second order polynomial.

Fig.3 shows p and the statistical differences between the measured and estimated ones. It is observed that in the first region, as the number of sockets increases, the packet loss increases only slightly. At some point, however, there is a knee in the curve where congestion effects begin to significantly affect the packet loss rate. TCP interprets packet loss as an explicit congestion notification from the network that indicates that the sender should decrease its rate of transmission.

The ability to predict p would provide a mechanism for big data environments to place an accurate commodity value on available network bandwidth for purposes of trading network bandwidth on an open Cloud Computing trading market. There is a trade-off between the sophistication of the model and measurements needed to fit it. An empirical formula with a confidence coefficient, R, for Packet Loss Rate (the number of retransmitted packets divided by the total number of packets transmitted), p, is obtained in the form of a complete 2nd order polynomial:

$$\mathrm{p} = \alpha \cdot \mathrm{n}^2 + \beta \cdot \mathrm{n} + \gamma \qquad (1)$$

Here *α*, *β* and *γ* are parameters to be fit based on measurements.

Any application using parallel TCP connections must select the appropriate number of sockets that will maximize throughput while avoiding congestion. Over the lifetime of a connection, bandwidth and delay change (due to transitory queuing, congestion and route changes, etc) imply that the bandwidth-delay product (BDP) of the connection also changes. Our tests of the BDP between Cambridge and Beijing at 10 minute intervals support this claim. nettimer [22] is used to measure dynamic latency and static bottleneck bandwidth. The bottleneck bandwidth averages 82.1 Mbps with a low and a high of 46.2 Kbps and 97.5 Mbps, respectively. The RTT delay also varies between (379, 687) ms with an average delay of 501 ms. As a result, the BDP for our connection varies by as much as 35 Mb.

Because the BDP over the lifetime of a connection is continually changing, a fixed value for n is not ideal. Selecting a fixed value forces an implicit decision between (1) under-allocating memory and under-utilizing the network or (2) over-allocating memory and wasting system resources. These fixed values are inappropriate even when the BDP is determined at the start of a connection since the BDP varies widely, even over short time scales, in wide area networks. In principle, the number of TCP flows can be manually tuned to fully utilize the under-utilized bandwidth, but this is a tedious process. Clearly, the big data community needs a solution that dynamically and transparently adapts n to achieve good performance without wasting network or memory resources. Dynamic Numbering Optimizer (DNO) is one such solution.

The optimal number of TCP streams, n, should satisfy Equation (2) [23]:

$$\frac{W_{max}}{RTT} = \frac{1}{RTT\sqrt{\frac{2bp}{3}} + T_0 \min\left[1, 3\sqrt{\frac{3bp}{8}}\right] p(1+32p^2)} \qquad (2)$$

In this equation, W_{max} is the maximum congestion window size, RTT is the round trip time, b is the number of packets of transmitted data that is acknowledged by one acknowledgement (ACK) from the receiver (usually b = 2), and T_0 is the timeout value. The above theoretical model was validated by a series of experiments [11]. It was shown that in the absence of congestion, the use of parallel TCP connections is equivalent to using a large MSS (Maximum Segment Size) on a single connection, with the added benefit of reducing the negative effects of random packet loss.

However, it is impossible to find out an explicit analytical formula for n since the order of the polynomial in Equation (2) is as high as 7th against n. A simple method is adopted: the knee in the aggregate throughput curve, determined by an optimal n, can be found by enumerating n from 1 to 128 because n can only take integer values. As shown in our experiments, the performance was steadily improving for up to 128 simultaneous streams; over 128 streams, instabilities and lower transfer rates were experienced. The RTT even varies during the lifetime of a connection. Clearly, dynamically adapting n can achieve good performance without wasting network or memory resources.

3.4 Fast-Recovery Effect of Multi-Streams

Slow-start is part of the congestion control strategy used by many network applications [24]. Slow-start is used to avoid sending more data than the network is capable of transmitting, that is, network congestion. When a loss event occurs every 1/p packets, the congestion window will be reduced by half. This leads to the classic "saw tooth" pattern. If we combine the two streams into the aggregate representation it is clear that the effect of using multiple network sockets is in essence equivalent to increasing the rate of recovery from a loss event by a factor of two. As the number of simultaneous TCP connections increases, the overall rate of recovery will increase until the aggregate network load begins to congest the network. At this point the TCP sender should reduce its congestion window.

Given that the aggregate rate of congestion recovery across all of the parallel TCP streams is functionally equivalent to an increased recovery rate, there is an interesting observation that can be made. TCP connections over wide area networks suffer from the disadvantage of long round trip times relative to other TCP connections that may have smaller round trip times. This disadvantage allows TCP senders with small RTTs to recover faster from congestion and packet loss events than TCP sessions with longer RTTs. Since the use of parallel TCP sockets provides a higher recovery rate, host with longer RTTs are able to compete on a fairer basis with small RTT TCP connections for bandwidth in the presence of congestion in the network bottleneck.

3.5 Multi-path Routing increases the aggregate bandwidth

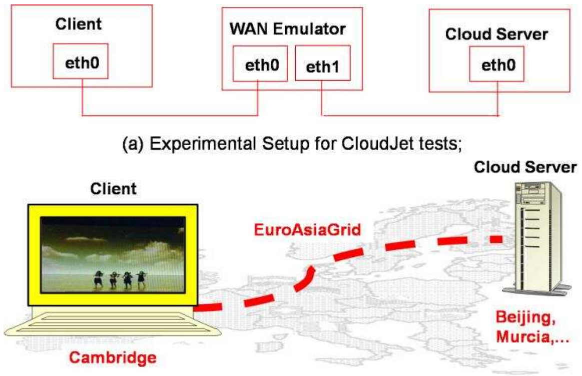
Parallel streaming data access also enables congestion-aware multi-path routing by taking advantage of path diversity [7][25][26][20]. Most of today's network routing protocols select only a single path between the source and the destination (an end-to-end link), limiting the achievable throughput. Using minimal congestion feedback signals from the routers, the flow at the source can be optimally split between each source-destination pair. The aggregate bandwidth of such multi-path routing should be theoretically enormous as long as the bottleneck is not formed at the two local connections to the Internet/clouds.

Parallel streams over a single path (route) may not be friendly to other users, allowing a single user to take a disproportionate share of available bandwidth. However, multipath routing [7][20] takes advantage of multiple paths (routes) between the source and the destination, therefore achieving theoretically unlimited aggregate bandwidth. Using minimal congestion feedback signals from the routers [7][20][25][26], the flow at the source can be optimally split between each source-destination pair. Furthermore, multicore processors [27] are ideal to fill this enormous bandwidth gap because they allow many users to connect to a site simultaneously through independent threads of execution. This enables Database servers (MySQL /IBM DB2), Web servers and application servers to have better throughput. The work of making the multiple connections go through different routes is still underway at our lab, including the utilization of tools such as traceroute to show actual routing multiplicity.

IV. EXPERIMENTS & MEASUREMENTS

The following real-world tests demonstrate the performance of the CloudJet code (alpha release) over the EuroAsiaGrid network resources [28], as shown in Fig.4. The EuroAsiaGrid Project is funded by the European Commission (EC) and six sites (Cambridge, London, Paris, Murcia, Delhi and Beijing) have been participating in the tests. At each site a Linux/Globus machine with installed CloudJet client gateway was used as a client to access a dedicated server with installed CloudJet server gateway at the Cambridge-Cranfield High Performance Computing Facility (CCHPCF) in the investigation. The CCHPCF has a 155 Mbps permanent connection to the Internet using 10 Gbps SuperJANET backbone via the EastNet Cambridge node, while at the time of the investigation Murcia had a 10 Mbps connection, London had a 2 Mbps connection, Beijing had a 100 Mbps connection. The Cambridge-Beijing datalink represents the longest network connection (geographic distance 10,000 km, 27 router hops, average RTT = 539 ms).

Some data transport mechanism was benchmarked across the emulated WAN, producing user-settable delay and packet drop probability (Fig.4(a)). The average round-trip time (RTT) ranged from 0 ms to 120 ms. The amount of real memory used by each of the machine types was as follows: Dell: 512 MB (client machines); Compaq: 256 MB (client machines); GOS as storage servers; IBM eServer 306, 512 MB. The TCP buffer size was 64 kB, the OS default value.



(a) Experimental Setup for CloudJet tests;

(b) Real-world CloudJet tests over EuroAsiaGrid.

Fig.4 CloudJet experimental setup.

4.1 Benchmarking CloudJet

Fig.5 measures the CloudJet's bandwidth as a function of RTT and the number of sockets (SockJ/#sock) [29][30][31]. Even in a LAN (Local Area Network) environment where RTT=0, and therefore the WAN emulator does not impose any time delay between the client and the servers, CloudJet still outperforms conventional TCP/IP. This is because the CloudJet protocol would perform multiple socket/TCP communication by taking advantage of the time used solely by the hardware and OS implementation in processing the received data and clearing the receiver buffer. When RTT = 40 ms, the maximum bandwidth increases to 69.3 MB/s with 8 sockets; when RTT > 40 ms, increases in bandwidth with further increased #sock can still be seen toward the right side of the graph. It seems that the bandwidth loss due to remote access can be simply retrieved, more or less, by increasing the number of sockets. However, when RTT = 40 ms, the maximum bandwidth decreases from 69.3 MB/s (8 sockets) to 65.9 MB/s with 16 sockets. It shows that CloudJet would eventually consume too much time in managing so many sockets, slowing down such an improvement.

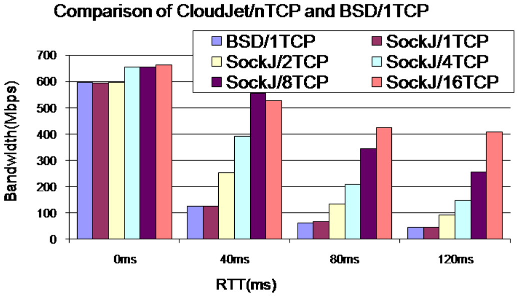


Fig.5 Storage-networking CloudJet's bandwidth as a function of RTT and the number of sockets (SockJ/#sock).

4.2 Accelerating distributed applications over CloudJet

As illustrated in Fig.4, OpenOffice [32], MySQL [33], IBM DB2 [34], Firefox browser [35], Media Player [36], Google Earth [37] are deployed on top of CloudJet. The detailed test parameters are listed in Table 1 [38][39][40][41]. These applications achieve up to tenfold accelerations, as summarized in Table 1. CloudJet addresses the challenge of sharing large volume data in the clouds with "LAN-like" performance, which is a highly beneficial aspect to the big data community. With the accelerated CloudJet protocol that remains fully compatible with the existing IT infrastructures, users can efficiently manipulate and move remote files and multimedia clips in the computational cloudss.

Table 1 CloudJet accelerates other distributed applications.

Applications	Performance over TCP/IP	Performance over CloudJet (#socket)	Speedup (max)
vi/Emacs	1241 s	119 s (#sock = 16)	10.4
	Opening a 16MB document remotely over the EuroAsiaGrid with a 4Mbps link to the Internet		
OpenOffice Writer/Calc/ Impress	189 s	29 s (#sock = 8)	6.5
	Saving a 16 MB modified document remotely over the EuroAsiaGrid with a 2 x 100Mbps link to the Internet		
MySQL	24101 s	4092 s (#sock=16)	5.9
	Backing up a 652 MB genome database via mysqldump over the EuroAsiaGrid		
Firefox	177 s	33 s (#sock=16)	5.4
	Downloading a 16 MB hyperlinked object from a remote Website		
MPlayer	15.1 frames/s	26.7 frames/s	1.8
	Playing a 36.4 MB Bondgirls video clip online from a remote Website		
Google Earth	409 s	54 s (#sock=16)	7.6
	Load a 41.8 MB map into a layer of the Google Earth browser		

V. CONCLUSIONS

Big Data results in connectivity needs for clusters of servers numbering in hundreds of nodes, which is often characterized by networks with large bandwidth-delay products. Today's data communication protocols are conservatively designed in terms of using only a small fraction of available network/disk bandwidth. The gap is widening between the maximum-achievable network bandwidth and the actually-utilized bandwidth. CloudJet is specially designed to solve this problem. Conforming to the universal VFS or BSD Socket interface, CloudJet can be pervasively used as an underlying platform to accelerate typical big data applications, including Data Mining, MySQL and other Office/Web/Media applications.

As shown in Fig.6 (Evolution of storage architecture for networks), to deploy storage resources on the network, one can choose to "split" different components in the complete application/data path. A NAS (Network-attached Storage) splits its filesystem via the NFS protocol (server/client mode). A NBD (Network Block Device) splits its device driver, making a remote disk on a different machine act as though it were a local disk on the local machine appearing as /dev/nda via a pair of split device drivers. An iSCSI splits its SCSI bus, allowing a machine to use an iSCSI initiator to connect to remote targets such as disks and tape drives on an IP network for block level I/O. As a variant or successor of NAS, a Grid-oriented Storage (GOS) appliance splits its specific GOS-FS protocol. Storage-networking protocol CloudJet in Cloud Storage conforms to POSIX and thereby can be pervasively used to accelerate any POSIX-compatible application in the clouds. Distributed applications such as distributed data mining can be accelerated by a factor of 2 - 10 in real-world CloudJet tests.

CloudJet conforms to the POSIX (Portable Operating System Interface) interface and pervasively speeds up nearly all network applications. This technique is featured with long distance and large volume. That is to say, due to the overhead of managing multiple threads, it does not show superiority over traditional protocols for short-distance or small volume storage-networking. It is a storage-networking protocol for big data rather than a new communication protocol in general.

CloudJet is a Socket-level interface that efficiently uses the existing data communication protocols. A CloudJet/Multi-path TCP combination can achieve theoretically unlimited aggregate bandwidth by taking advantage of multiple routes between the source and the destination. This is friendly to other users because a single user does not need to take a disproportionate share of available bandwidth from a single path.

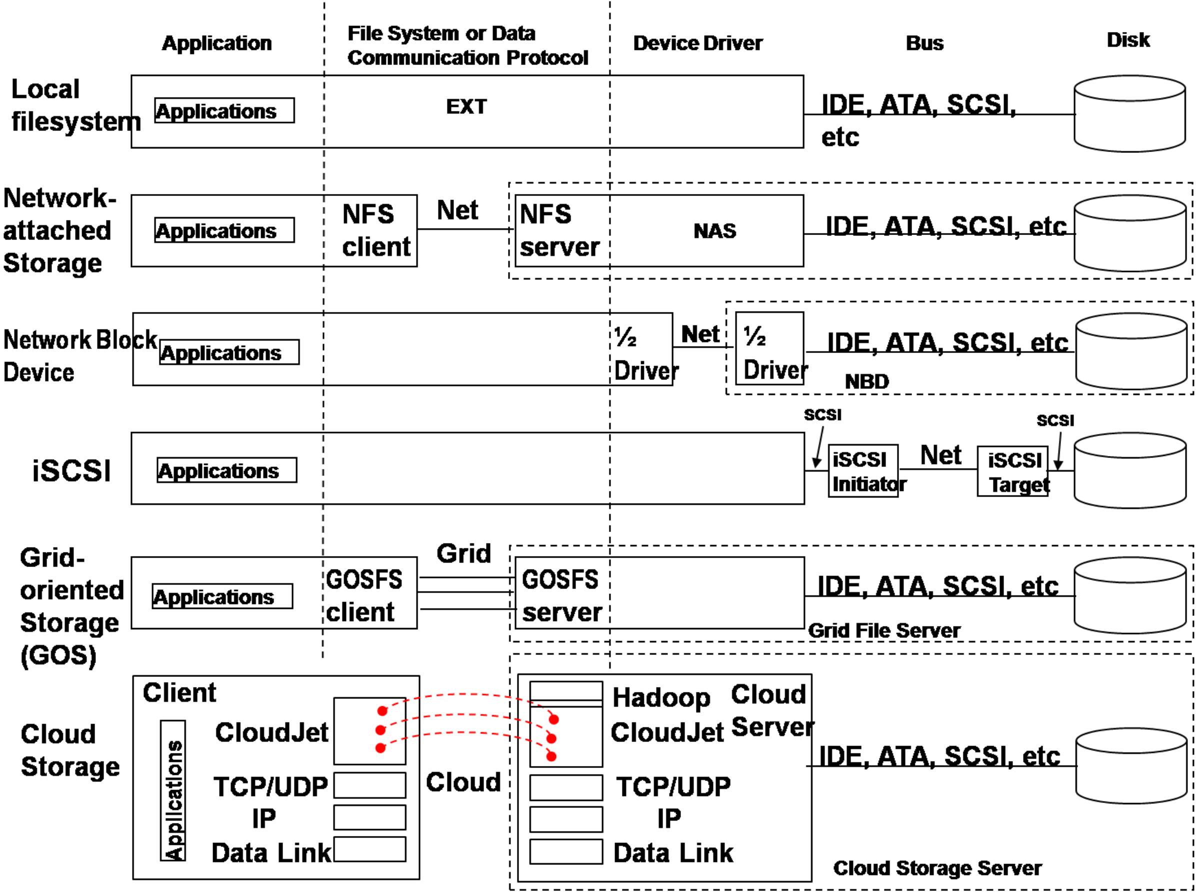


Fig.6 Storage architecture for networks continues to evolve.

Acknowledgements

We thank Prof Garth Gibson of Carnegie Mellon University, Prof Kai Li of Princeton University, Prof Ric Parker of Rolls Royce, Dr Richard Wright of BBC, Dr. Flavia Donno of CERN, Dr. Sophie Vandebroek of Xerox, Dr. Paddy Francis of EADS, and Prof Lionel Ni of HKUST for viewing our demonstrations and providing us with comments that much improved the CloudJet work. This work is sponsored by the UK government and European Commission (EC) through an EPSRC/DTI grant (£ 1 million) “Grid-oriented Storage (GOS)”, an EPSRC grant (£470k) "Accelerating NFS/CIFS", an EC grant (€1 million) “QuickLinux” and an EC grant (€400k) “EuroAsiaGrid”.

References

[1] M. Armbrust, “Above the Clouds: A Berkeley View of Cloud Computing”, UC Berkeley Reliable Adaptive Distributed Systems Lab, 2009.

[2] D. Wetherall & A. Tanenbaum, Computer Networks. Upper Saddle River, NJ: Pearson Prentice Hall. ISBN 0-13-212695-8, 2011.

[3] B. Tleney, L. Tierney, Dan Gunter, Jason Lee, Martin Stoufer & Joseph B. Evans, “Enabling Network-Aware Applications”, proceedings of the Tenth IEEE International Symposium on High Performance Distributed Computing (HPDC), August, 2001, San Francisco, CA, 2001

[4] E. Kosar, “Balancing TCP Buffer Size vs Parallel Streams in Application-Level Throughput Optimization”, The Second International Workshop on Data-Aware Distributed Computing, Munich, Germany - June 9, 2009

[5] B. Pearce, Breaking Moore's law, A brief history of the Grid. London: GridCafé.

[6] J. Leake, Coming soon: superfast internet. TimesOnline.

[7] H. Han, “Multi-Path TCP.” IEEE/ACM Transactions on Networking 14, no. 6: 1260-1271., 2006

[8] J. Toigo, The holy grail of data storage management. Prentice Hall, 2000

[9] S. Plank, Micah Beck, Wael Elwasif, Terry Moore, Martin Swany, Rich Wolski, Internet Backplane Protocol: Storage in the Network, 2004

[10] M. Li, J. Shu, W. Zheng, “GRID codes: Strip-based erasure codes with high fault tolerance for storage systems.” ACM Transactions on Storage 4, no.4: 15, 2009

[11] T. Hacker, “The Effects of Systemic Packet Loss on Aggregate TCP Flows.” Proceedings of the international conference on Supercomputing. ACM, 2002

[12] S. Anastasiadis, R. Wickremesinghe, J. Chase, “Rethinking FTP: Aggressive block reordering for large file transfers.” ACM Transactions on Storage 4, no. 4: 13, 2009

[13] F. Wang, S. Wu, N. Helian, Y. Deng, A. Parker, Y. Guo, V. Khare, “Grid-oriented Storage: A Single-Image, Cross-Domain, High-Bandwidth Architecture.” IEEE Transaction on Computers 56, no. 4, 2007

[14] F. Wang, et al, “Grid-oriented Storage: Parallel Streaming Data Access to Accelerate Distributed Bioinformatics Data Mining.” ACM/IEEE SuperComputing Storage Challenge Finalist, USA, 2007

[15] M. Fisk, “Dynamic Right-Sizing: TCP Flow-Control.” ACM/IEEE Super Computing, 2001

[16] I. Foster, GridFTP Documentation. Chicago: Global Grid Forum, 2009

[17] M. Jones, etter networking with SCTP--The Stream Control Transmission Protocol combines advantages from both TCP and UDP. IBM, 2006

[18] D. Wei, IEEE/ACM Trans. on Networking 19, no. 1: 4-11, 2007

[19] T. Kelly, “Scalable TCP: improving performance in highspeed wide area networks.” ACM SIGCOMM Computer Communication Review 32, no. 2: 83-91, 2003

[20] D. Wischik, C. Raiciu, A. Greenhalgh, M. Handley, NSDI '11: 8th USENIX Symposium on Networked Systems Design and Implementation, Boston, 30 March 2011.

[21] D. Kostic, Adolfo Rodriguez, Jeannie Albrecht, Amin Vahdat, Bullet: High Bandwidth Data Dissemination Using an Overlay Mesh," Proceedings of the 19th ACM Symposium on Operating System Principles (SOSP 2003), October 2003.

[22] K. Baker, Nettimer: A Tool for Measuring Bottleneck Link Bandwidth. Stanford University, 2009

[23] T. Hacker, B. Athey, “The End-to-End Performance Effects of Parallel TCP Sockets on a Lossy Wide-Area Network”, Proceedings of the 16th International Parallel and Distributed Processing Symposium, 2001

[24] E. Cohen, “Managing TCP Connections under Persistent HTTP.” The International Journal of Computer and Telecommunications Networking 31, no. 11-16: 1709-1723, 1999

[25] C. Argos, “Adaptive Multi-Path Routing for OBS Networks.” 9th International Conference on Transparent Optical Networks, 2007

[26] C. Raiciu, Sebastien Barre, Christopher Pluntke, Adam Greenhalgh, Damon Wischik, and Mark Handley, Improving datacenter performance and robustness with multipath TCP. In Proceedings of the ACM SIGCOMM conference (SIGCOMM '11). ACM, New York, 2011

[27] C. Leiserson, How to Survive the Multicore Revolution. Cilk Arts., 2009

[28] CCHPCF, The EC FP7 EuroAsiaGrid/QuickLinux Project, Cambridge-Cranfield High Performance Computing Facility (CCHPCF), www.hpcf.cam.ac.uk, 2010

[29] B. Martin, Using Bonnie++ for filesystem performance benchmarking. Linux, 2008

[30] F. Guo, “From chaos to QoS: case studies in CMP resource management.” ACM SIGARCH Computer Architecture News 35, no. 1: 21-30, 2007

[31] INSDC, “The International Nucleotide Sequence Database Collaboration.” http://insdc.org, 2010

[32] M. Muller-Prove, “Community experience at OpenOffice.org.” ACM Interactions 14, no. 6: 47-48, 2007

[33] MySQL 2012. “MySQLQuery Analyzer - Improving SQL Query Performance.” www.mysql.com.

[34] D. Zilio, “Self-managing technology in IBM DB2 universal database.” Proceedings of the tenth international conference on Information and knowledge management. ACM, 2001

[35] Firefox web browser: Faster, more secure, & customizable, www.mozilla.com/firefox, 2012

[36] R. Urunela, “Energy adaptation for multimedia information kiosks.” Proceedings of the 6th ACM & IEEE International conference on Embedded software. ACM, 2006

[37] C. Schulze, “Sketch-based annotations in Google Earth.” ACM SIGGRAPH, 2009

[38] D. Kotz, “Disk-directed I/O for MIMD multiprocessors.” ACM Transactions on Computer Systems 15, no. 1: 41-74, 1997

[39] M. Zhao, “A user-level secure grid file system” Proceedings of the 2007 ACM/IEEE conference on Supercomputing. ACM, 2007.

[40] D. Ferraiolo, “Proposed NIST standard for role-based access control.” ACM Transactions on Information and System Security 4, no. 3: 224-274, 2001

[41] J. Liu, “Virtualization polling engine (VPE): using dedicated CPU cores to accelerate I/O virtualization.” Proceedings of the 23rd international conference on Supercomputing. ACM, 2009