Impact of Information Security measures on the Velocity of Big Data Infrastructures

Lionel Dupré
EBRC (European Business Reliance Centre)
Luxembourg
lionel.dupre@gmail.com

Yuri Demchenko
University of Amsterdam
y.demchenko@uva.nl

Abstract— Encryption is often viewed as a major drawback which hinders the performance of processing systems. This perception is not wrong; encrypted storage, memory and communications usually perform much slower than systems which process data in the clear. Big Data applications is no exception to the rule: it was designed with Volume and Velocity requirements in mind, and security (i.e. encryption) was initially not considered; perimeter security was deemed sufficient, and Big Data systems were confined to back-end operations. Considering the recent developments in the field (AES-NI processors, Key Management Servers, homomorphic encryption), the encryption vs performance paradigm needs to be actually measured to re-evaluate preconceived reservation.

This research found that encryption is no longer an obstacle to efficient and fast Big Data processing, thanks to massive processing parallelisation (which distributes also the encryption payload), new CPU technologies which allow encryption instructions to perform much faster, the use of SSD storage, and finally the clever data-centric use of encryption in HDFS. The paper provides analysis of four strategies in using data encryption in Hadoop based Big Data applications, which have been tested on the testbed built on Amazon Web Services (AWS) platform using advanced AWS monitoring data. Tests were performed on datasets of relatively modest size (about 5-20 Gigabytes), and performance was measured as all data could fit in each node's RAM. On larger datasets (e.g. of Terabytes scale), data partitioning may be required to obtain similar results.

***Index Terms*— Big Data Infrastructure, Big Data Security, Data Encryption, Hadoop, Big Data Applications Performance.**

I. INTRODUCTION

Big Data technologies are widely used by business and research. The most recent works of the NIST Big Data Working Group report that Big Data adoption now concerns more sectors than just academics or research; many critical sectors such as health or finance now take advantage of massively parallelised data storage and processing [1]. In those sectors, the regulatory landscape imposes that many requirements are fulfilled without reserve. HIPAA or Financial regulations impose strict liability to companies which require as a consequence that both systems and processed data are adequately protected and can be used as proof, e.g. in case of legal dispute or forensics investigation [2]. As a reflex, such companies would most likely enforce perimeter security or choose a solution which is deemed "secure". Perimeter security however doesn't address compliance risks, nor other growing risks such as insider threats, or information leakage [2][3][4][5].

Hadoop was however initially not built with Security concerns in mind [6]; the prime objectives were the generic 3 Vs properties (Volume, Velocity, Variety) which are today complemented by other properties (Veracity, Variability, Value). Controls were only bound to address errors (e.g. data loss, disk corruption, etc.) but not malicious usage. Recent research envisage encryption (i.e. data-centric security) as a the most comprehensive way to embrace security needs in Hadoop on top of classic network, servers and applications security without implementing major architecture changes [2]. Privacy and regulatory compliance indirectly require such data-level protection.

The remainder of the paper is organised as follows. Section II describes the core components of Hadoop based Big Data Infrastructures (HBDI), Section III proposes Security Strategies and possible solutions, Section IV describes the testbed that was designed to assess the impact of Security on the HBDI Velocity, Section V exposes the findings and Section VI provides an analysis of the results.

II. CORE COMPONENTS OF BIG DATA INFRASTRUCTURE (BDI)

Fig. 1provides a general view on the Big Data infrastructure as a part of the Big Data Architecture Framework (BDAF) defined in the authors' previous work [7] that includes the general infrastructure for general data management, typically cloud based, and Big Data Analytics infrastructure that will require specialised and high-performance computing clusters. General BDI services and components include:

- Big Data Management tools
- Registries, indexing/search, semantics, namespaces
- Security infrastructure (access control, policy enforcement, confidentiality, trust, availability, privacy)
- Collaborative environment (groups management)
- The Federated Access and Delivery Infrastructure (FADI) as an important component of the general BDI that interconnects different components of the cloud/Intercloud based infrastructure combining dedicated network connectivity provisioning and federated access control.

Besides the general cloud base infrastructure services (storage, compute, infrastructure/VM management) the following specific applications and services are required to support Big Data and other data centric applications:

- Hadoop based services and tools, streaming analytics, etc.
- Specialist data analytics tools (events, data mining, etc.)
- Databases/Servers SQL, NoSQL

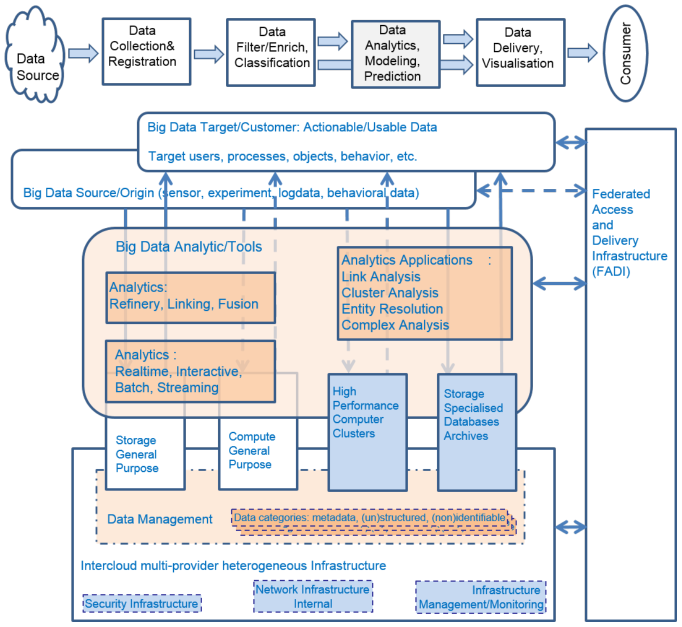

Data Source
Data Collection& Registration
Data Filter/Enrich, Classification
Data Analytics, Modeling, Prediction
Data Delivery, Visualisation
Consumer
Big Data Target/Customer: Actionable/Usable Data
Target users, processes, objects, behavior, etc.
Big Data Source/Origin (sensor, experiment, logdata, behavioral data)
Federated Access and Delivery Infrastructure (FADI)
Big Data Analytic/Tools
Analytics: Refinery, Linking, Fusion
Analytics Applications : Link Analysis Cluster Analysis Entity Resolution Complex Analysis
Analytics : Realtime, Interactive, Batch, Streaming
High Performance Computer Clusters
Storage Specialised Databases Archives
Storage General Purpose
Compute General Purpose
Data Management
Data categories: metadata, (un)structured, (non)identifiable
Intercloud multi-provider heterogeneous Infrastructure
Security Infrastructure
Network Infrastructure Internal
Infrastructure Management/Monitoring

Fig. 1. General Big Data Infrastructure functional components

A. *Security Strategies for Big Data*

The relative complexity of the software architecture of a BDI increases the difficulty of the challenge for implementing end-to-end security. Confidentiality and integrity requirements are however increasingly demanded; and one of the key requirements for improving security is to ensure that it is applied thoroughly. In the minds of many security experts, security architectures must still be typically system-centric and one forgets easily what we have to protect data in the first place: system-centric security (and equally physical security) are of course a prerequisite, but the holy grail of security is rather information-centric, driven by privacy and compliance constraints.

Encryption is a key element of an adequate security strategy, but it introduces several obstacles and challenges that are difficult to overcome: Considering the typical architecture of a HBDI (see Fig. 2), several approaches can be engineered to protect either the system, the data, or both. We studied the pros and cons of four different strategies.

1) *Strategy #1: Inter-regions encrypted tunnels and VPCs*

A "simplistic" solution consists in reusing the Virtual Private Cloud (VPC) concept promoted by Amazon Web Services (AWS) or its competitors: instead of securing down to each communication channel, encryption only occurs between two nodes if they are located in different regions. Hadoop components would therefore live inside "bubbles" which are interconnected using encrypted tunnelling: typically, the risks of man-in-the-middle or eavesdropping attacks are reduced.

2) *Strategy #2: Non-Hadoop security components*

Another interesting option consists in using encryption mechanisms built-in the operating system [8]. For data in transit, IPsec (transport layer) is a fair choice: it is standard, available on all kinds of OS, and often open-source. We could therefore force all traffic (ingress/egress) to pass through an IPsec tunnel for all communications. data at rest could be encrypted by the Operating System (OS).

3) *Strategy #3: In-Product security components and Hadoop "Secure Mode"*

As we saw in Strategy #2, we still need to cover the risk of insider threat to ensure that data is properly compartmented and that compartments are entirely imperviously sealed. For this purpose, we propose both AAAA (authentication, authorization, accounting and auditing) mechanisms [9], encryption of data in transit, and encryption of data at rest performed in a way so that encryption is linked to a credentials pair, and not to the machine (unlike whole disk encryption).

For all RPC traffic, Hadoop implements an SASL authentication layer which has also data in transit encryption abilities [10]; RPC/SASL configuration must however be maintained carefully and regularly during the system's lifecycle as compatibility fall-back mechanisms allow a node to challenge/response and negotiate little or no encryption at all. Client connectivity would be secured using HTTPS and TLS: this is similar to all websites security currently used. Here, a Man-in-the-middle attack would target the root CA, or the delegates root CAs, since TLS is most often not used for mutual authentication (i.e. when client and server mutually authenticate). Such mutual authentication could be however highly desirable.

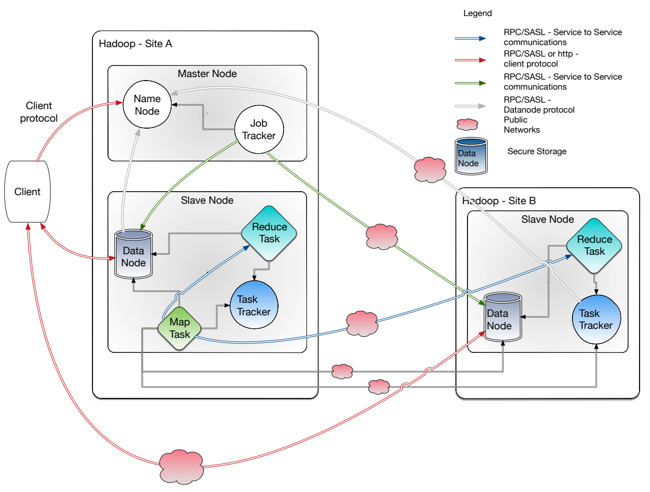


Fig. 2. Per protocol encapsulation / encryption add-on

4) *Strategy #4: HDFS transparent encryption*

As the implementation of server-side encryption in Hadoop still represents a major challenge, one alternative would consist in implementing end-to-end encryption. Hadoop already proposes such feature as part of its own software distribution [11]. It relies on a Key Management System (KMS) as shown in Fig. 3. The KMS is the encryption keys custodian; it can authenticate an account and assign to each CRUD permissions

(Create, Read, Update and Delete). HDFS must also offer the capability to define mandatory encryption zones/containers where data cannot be stored in the clear. The data stored in HDFS encrypted zones must also be accessible by processes (e.g. during the MapReduce phase). The KMS must therefore be summoned equally by a Client when data is stored or read, and by the Name Node.

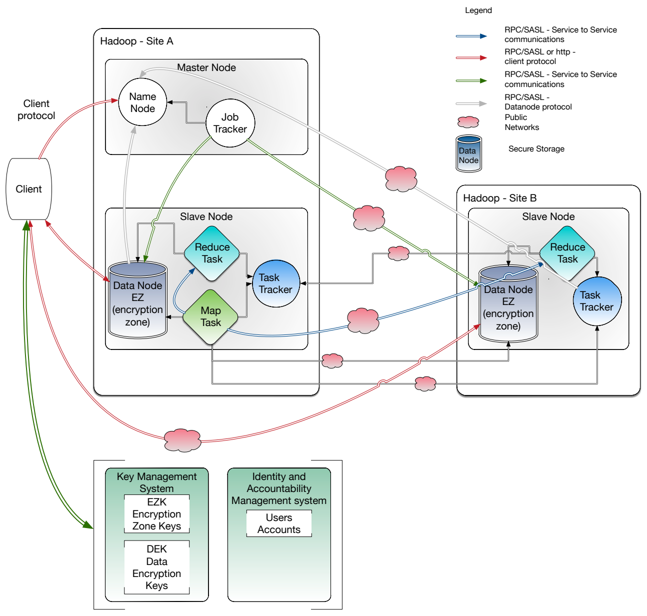


Fig. 3. HDFS transparent encryption

B. *Strategies drawbacks*

1) Strategy #1 (Inter-regions encrypted tunnels and VPCs)

A machine to machine encryption seems at first a very attractive solution as its deployment could be almost fully automated. Such setup however presents several drawbacks:

- Resizing of the cluster could prove to be difficult or compromise the symmetric cryptographic key used.
- The solution does not mitigate the risk of Insider Threat at all; once an SSH / VPN access is established, an 'authorised' person could still poison data, or destroy it.

2) Strategy #2 (Non-Hadoop security components)

The reliance on solely external security mechanisms is rather cumbersome to implement, and could also leave many vulnerabilities open. For instance, the correlation of logs -and therefore the overall system accountability as requested by HIPAA- would be far from achieved.

3) Strategy #3 (In-Product security components and Hadoop "Secure Mode")

While node to node authentication is highly desirable to ensure no rogue HBDI component takes part to the cluster, a number of compensating controls are required to provide thorough data assurance. Here again, we would lack native correlation of logs among security components, for instance.

4) Strategy #4 (HDFS transparent encryption)

Such an approach is interesting, as it ensures that:

- Data in transit (between the client and the HDFS storage) is encrypted;
- Data at rest (in HDFS containers) remains encrypted.

The model is far better than the others; it however creates a strong availability dependency (and possibly a significant performance impact) on Key Management System's (KMS) availability. Furthermore, a number of vulnerabilities remain open and require mitigation where possible:

- Hardware access exploits: same as other solutions. The attacker would however obtain access to encrypted files on disk.
- Root access: a memory dump during processing would still allow to reveal data. This is also the case in other setups and no mitigation could possibly exist (unless when using anti-tampering hardware).
- Insider threat: someone who would manage to steal an HDFS account could access the KMS, and the associated keys and run processes to dump data to external storage. This cannot be mitigated.

Additional measures are still required:

- SASL protocol is required to protect node-to-node and user-to-node authentication and accountability.
- The initial data upload would still need to occur over an HTTPS link.
- Multi-Tenant isolation would require to be strictly enforced to prevent potentially Denial of Service on HBDIs' components [11].

III. Experimental Testbed

Out of the four strategies studied in theory, only Strategy #4 seems to be realistic. The first three present serious drawbacks – and could be often subject to serious implementation limitations– and we have therefore chosen to implement only Strategy #4. Our experience used tools and platforms available to non-Experts, and the purpose is to evaluate how encryption practically impacts performance. For our purpose, we used a commercially available Big Data offer, and implemented a reusable testbed (see Fig. 4) on AWS.

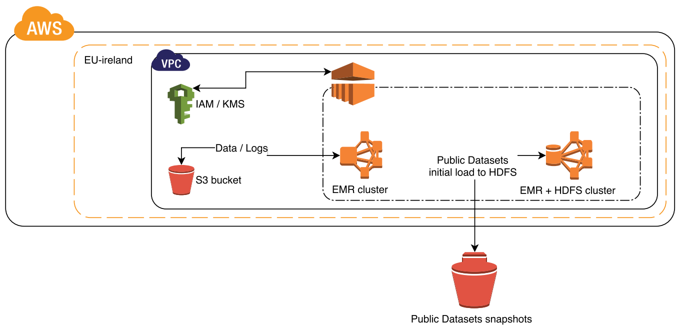


Fig. 4. initial architecture for the Testbed using AWS free tier

The testbed is composed of several AWS services: S3 (simple storage service) buckets (AWS's storage service), an IAM (Identity and Accountability Management system), and an EMR (Elastic Map Reduce) cluster of nodes. EMR is the commercial AWS name for its own Hadoop implementation; which means that we can either use it the way AWS set it up by default (e.g. using S3 buckets), or use it as a typical HBDI using its core HDFS storage. In EMR, HDFS storage is not persistent, but this is not required by our tests at this stage. We created 4 different setups on which we could run several times identical jobs with 2, 4, 8, 16 and ultimately 20 machines (AWS's limitation), and produced measurement sets that we compared.

A. *Hardware configuration*

In order to remain consistent with the aim of the experiment (an applied testing of encryption impact on performance), we selected a standard hardware configuration which is proposed by default EMR/EC2 instances. The hardware configuration of these instances is as follows:

- 2.6GHz Intel Xeon E5-2670 v2 (Ivy Bridge) Processors (which include an AES-NI module)
- 2x40GB SSD-based instance storage
- 4 vCPUs (1 CPU, 4 core)
- 15Gb RAM.

B. *Data processing preparation*

1) *Sample Dataset*

We selected a dataset of a medium size, already public or considered public, and available in a format that could not hinder the production of results. The choice was rapidly set on a database dump of Wikimedia products:

- the Wiktionary English XML monthly dump (4.3GB, b2zip format, without images)
- the DBpedia English dump (17GB, b2zip format, without images).

2) *Text Processing using a PIG script launched from the GRUNT shell*

The processing of such datasets therefore indeed qualify as "Big Data", since they "cannot be processed in Microsoft Excel". A script was required; the choice of PIG (present in all Hadoop standard installations, without further add-ons) was made obviously because of its simplicity. The development of the script (based on several examples available in online tutorials) produced this final version which was kept simple to stay away from bottlenecks introduced by code complexity:

```
lines = LOAD '$INPUT' USING TextLoader
AS (line:chararray);

words = FOREACH lines GENERATE
FLATTEN(TOKENIZE(line)) AS word;

grouped = GROUP words BY word;

wordcount = FOREACH grouped GENERATE
group, COUNT(words);

STORE wordcount INTO
'$OUTPUT/wordcount.txt';
```

Fig. 5. PIG wordcount script

C. *Configuration aspects of the Testbed*

1) *Step 1: no security testbed*

During this phase, the testbed consisted in an EMR deployment of 1+2 (master+slave) nodes, with a core setup of Amazon Hadoop 2.6.0 + Hive 1.0.0 + Pig 0.14.0 + Ganglia 3.6.0 (default AWS EMR setup, December 2015). EMR relied on an initial setup using S3 buckets, AAA in place, no encryption.

2) *Step 2: EMRFS server-side encryption*

Step 2 is similar to step 1, but storage is EMRFS (i.e. enhanced and Hadoop-compliant S3 storage) with Server Side encryption using a key generated in AWS' KMS and using AES256 as the default encryption standard.

3) *Step 3: HDFS, no encryption*

During step 3, S3 buckets were only used as a repository to copy data to the HDFS storage which is provided with the EC2 (Elastic Compute Cloud, AWS's name for virtual machines) instances as part of the EMR configuration. Instead of using EMR's web console, we accessed the master node using SSH, importing the data into the EMR's HDFS area (which is not secure at this stage yet) and run the PIG script using a GRUNT shell.

4) *Step 4: HDFS transparent encryption*

Step 4 is similar to Step 3. We however use an encrypted HDFS zone to store both initial data and processing output. The encrypted zone was setup using HDFS cryptographic abilities. The KMS is the component provided by Hadoop and is independent from the AWS's IAM.

5) *IAM and KMS setup*

AWS's IAM (Identity and Access Management) product is a prerequisite to the setup of the EC2 instances which run the Hadoop distribution; they also do not link to the Hadoop's KMS in any way. They are however required if the chosen storage is S3 instead of HDFS.

D. *Metrics collection and analysis*

In order to compare the various Hadoop / EMR setups (non-secure, client-side encrypted, server-side encrypted), we need to pre-establish a set of metrics. AWS proposes such metrics collection in their product line (AWS's CloudWatch, which isn't available by default but can be setup instantly). Other products are also available, if the instances of the HBDI are self-operated or if more detailed metrics are required [13]. Since an EMR cluster is composed of 1/ EC2 instances, 2/ Temporary EC2 storage (temp files and swap), 3/ EMR and 4/ EMRFS (over S3) storage, we monitored metrics specific to each of these components:

- EC2 instances: CPUUtilization, NetworkIn, NetworkOut.
- Temporary EC2 storage (temp files and swap): DiskReadOps, DiskWriteOps, DiskReadBytes, DiskWriteBytes
- EMR: ClusterStatus, MapReduceStatus, NodeStatus, IO.

E. *Purpose of an Initial Dry Run*

The simplest testbed setup is initially required to establish a performance baseline which can be later compared to secure implementations. The lab therefore consisted in a succession of testbeds which configurations evolved in a similar way so that they can be later compared:

- 1+2 using S3 storage without encryption
- 1+2 using S3 storage with Server-Side AES256 encryption
- 1+2 using HDFS without encryption

- 1+2 using HDFS with transparent, AES256/CTR/NoPadding encryption.
- Repeat with 1+4
- Repeat with 1+8
- Repeat with 1+16
- Repeat with 1+19 (maximum of EC2 instances running is 20 overall).

IV. Results and Evaluation

A. *Medium-size dataset processing (4,7GB, flat)*

Our first objective consisted in the observation of the performance of the PIG script + the Wiktionary dump altogether and attempt to identify general, high-level conclusions that we could later explore further.

1) S3 bucket, no encryption

The initial run -which we qualify as "dry run" since it is not yet charged by any security mechanism (except basic AAAA, since it is a service enabled by default on AWS and consorts)- demonstrated that a roughly 4,7GB dataset can be processed rather rapidly using only one master and two slave nodes (1+2). Doubling the amount of slave nodes helped gaining significant processing time (approx. 50%), but to half the processing performance, we had to quadruple the 4 slave nodes. Adding a set of 3 additional nodes to the set of 16 which manages to process 4,7GB in 3:42 shows only little improvement, roughly 24s. Obviously, the horizontal upgrades tends to a limit which appears to be set around 3min for the processing of 4,7GB.

slave nodes #	1+2	1 + 4	1 + 8	1 + 16	1 + 19
processing time (s)	00:14:38	00:07:42	00:05:12	00:03:42	00:03:18

Fig. 6. 4,7GB, S3 buckets, no encryption

2) S3 bucket (EMRFS), server-side encryption (SSE) AES256, managed

Adding Server-Side Encryption (SSE) to S3 automatically forces EMR to implement the EMRFS "interpret" so Hadoop can access S3 (with SSE). Considering that Hadoop accesses a dataset which is not compressed, smaller than the size of the RAM available, we expect that the impact on performance should be hard to quantify precisely. Indeed, figures tend to show almost no difference (or even gains in some cases) of processing time compared to the use of S3 without encryption.

slave nodes #	1 + 2 + AES256	1 + 4 + AES256	1 + 8 + AES256	1 + 16 + AES256	1 + 19 + AES256
processing time (s)	00:14:14	00:07:56	00:05:12	00:03:26	00:03:06

Fig. 7. 4,7GB, S3 buckets (EMRFS), Server-Side encryption

3) S3 bucket - no encryption versus AES256

Overall, no performance impact appears consistently throughout the various steps of the experiment. For instance, in a 1+2 configuration, the setup with encryption performed 3% faster than without. No change was noticed in the 1+4 configuration. We even observe an 8% time gain in the 1+16 configuration, which does not verify our initial hypothesis that encryption would diminish performance by 4-10%, as supported by professional presentations and literature. Using such figures, we can deduct that:

- The encryption processing payload is also distributed across the nodes.
- The more slave nodes are used, the less an important dataset is loaded in memory, and the faster encryption / decryption will occur.
- We can suspect that encryption somehow forces the caching of the entire subset of data on each node, increasing the speed of processing.
- The m3.xlarge EC2 instances use 2.6GHz Intel Xeon E5-2670 v2 which implement the AES-NI (new instructions set)
- Each EC2 instance provides 15Gb of RAM and SSD drives which reportedly accelerate Hadoop processing by around 31% (Cloudera, 2015).

Considering these hypotheses, we can observe that SSE's impact on performance is not really quantifiable for a dataset of this size. In the table below, a negative performance impact means a gain of processing time after encryption is enabled.

slave nodes #	1+2	1 + 4	1 + 8	1 + 16	1 + 19
Encryption Performance Impact	-3%	3%	0%	-8%	-6%

Fig. 8. 4,7GB, S3 buckets (EMRFS), no encryption versus AES256 Server-Side encryption

4) HDFS, no encryption

Since we used an S3 bucket for our initial dry run, we used it as a baseline to compare the course of events when testing other mechanisms, including the native HDFS which is provided with Hadoop. HDFS is notably not a great performer, but has proven to be extremely reliable in ensuring data consistency and integrity thanks to its replication mechanisms, and the integrity checks it performs continuously. Overall, HDFS still provides a valuable service, despite the complaints we could read about its overall performance. Furthermore, HDFS was until Hadoop v2.6.0 unable to provide any security function to data. Since, HDFS transparent encryption was introduced and HDFS is now able to play a role in the security of the information it contains.

To start with, we have uploaded our dataset onto the Hadoop Master Node from an SSH shell, using the grunt shell's `cp` command. For the 4,7GB dataset, the `cp` command took a little below 1min to complete. Once aboard, the dataset was ready to be processed; we used the same script as previously; we however launch each line of the script using the grunt shell.

Performance metrics show that HDFS performs much better than S3 when only few slave nodes are used (up to 8). Above 8 slave nodes, HDFS seems to provide no further advantage and tends to a lower processing time limit of 4min, instead of 3min for S3.

slave nodes #	1+2	1 + 4	1 + 8	1 + 16	1 + 19
processing time (s)	00:10:47	00:06:03	00:04:33	00:04:13	00:04:17

Fig. 9. 4,7GB, HDFS data folder, no encryption

5) *HDFS, server-side encryption (SSE) AES256*

As shown in Fig. 3 (strategy #4), HDFS uses a native KMS (Key Management Server) to implement on-the-fly data encryption. We first implement an Encryption Zone (EZ), where HDFS forces each file to be encrypted [14]:

- Files are encrypted/decrypted using DEKs (Data Encryption Keys).
- DEKs are protected by being encrypted by the EZ key. The result is an EDEK.
- Users / applications may request a file to be decrypted: the KMS will serve the request only if the user is authorised.

As a result, users can be created, modified, deleted, and new users created; if one account is granted the right to access a file, it is actually granted a right to use an EDEK. This means that if a file is being transferred from a node to another, it will be under its encrypted form. Should SASL not be used between nodes, an MITM would fail to read the data itself. Such complexity worried us that performance impact could indeed be significant. The results however once again contradict the initial hypothesis.

slave nodes #	1 + 2 + AES256 CTR	1 + 4 + AES256 CTR	1 + 8 + AES256 CTR	1 + 16 + AES256 CTR	1 + 19 + AES256 CTR
processing time (s)	00:10:27	00:06:12	00:04:36	00:04:36	00:04:11

Fig. 10. 4,7GB, HDFS transparent encryption, Encrypted Zone dataE folder

6) *HDFS - no encryption versus AES256*

We can compare HDFS's performance with transparent encryption versus without encryption. Except a notable performance impact (8%) using the 1+16 configuration, other figures tend to show rather stable figures, like when using S3.

slave nodes #	1+2	1 + 4	1 + 8	1 + 16	1 + 19
Performance Impact	-3%	2%	1%	8%	-2%

Fig. 11. 4,7GB, HDFS, no encryption versus AES256 CTR transparent encryption

B. *Larger-size dataset processing (17GB, compressed)*

Previous experiments reveal that for "small" Big Data (i.e. 5GB), encryption has a very limited impact on performance. In some cases, it even forces caching which sends the entire file encrypted to RAM, where decryption performance is really high. We therefore wanted to use a file which "doesn't fit" in RAM, i.e. a 17GB, compressed text file, to increase the difficulty and observe whether encryption would be a strong challenge to the MapReduce processing performance.

To achieve this goal, we a) restored a snapshot to an EBS attached volume, b) mounted the EBS volume to the EMR Master Node, c) copied the content to HDFS storage (normal zone, and encrypted zone) and finally d) launched the same series of processes than previously, with the same horizontal scaling.

1) *HDFS, no encryption*

The size increase, combined with the use of b2zip compressed dataset is supposed to impact performance in an extremely heavy fashion, and allow us to make the part of encryption in the overall processing payload. The initial run shows indeed that for the size increase, the processing time increase is significant: in a 1+2 configuration, the 17GB compressed takes around 920% more time to process than the 5GB uncompressed. A 1+4 configuration (actually doubling the computing power) shows a halved processing time compared to 1+2, and 1+8 seems to tend already to the limit; 1+16 and 1+19 both take around 20min to complete the work.

slave nodes #	1 + 8	1 + 16	1 + 19
elapsed processing time (s)	00:28:09	00:25:29	00:25:19

Fig. 12. 17GB b2zipped, HDFS, no encryption

2) *HDFS, transparent encryption (AES256 CTR)*

The same tests using transparent encryption, i.e. loading the data in the EZ (encryption zone) which forces the processing to start after decryption has occurred and before encryption occurs again. We saw previously that encryption had little impact on performance on a dataset smaller than the amount of RAM available; with this compressed dataset, 17GB could mean that the overall size (uncompressed) would reach around 50GB (since bzip2 has a compression rate of 2,92. In all cases, the total data size is higher than the RAM on each EC2 instance, which may require optimisation and which explains the very poor performance figures in 1+2 and 1+4 configurations.

Unfortunately, we could not manage to complete the processing of the encrypted dataset in the 1+2 and 1+4 configurations: processing typically froze straight after 16% completion at each attempt.

slave nodes #	1 + 8 + AES256 CTR	1 + 16 + AES256 CTR	1 + 19 + AES256 CTR
elapsed processing time (s)	00:27:39	00:20:43	00:19:53

Fig. 13. 17GB b2zipped, HDFS, AES CTR

3) *HDFS - no encryption versus AES256 CTR*

On 1+8 and above setups, we can observe that encryption actually benefits the processing time by significant means: 23% gain in a 1+16 configuration, 27% gain in a 1+19 configuration.

slave nodes #	1 + 8	1 + 16	1 + 19
Performance Impact	-2%	-23%	-27%

Fig. 14. 17GB b2zipped, HDFS, AES CTR

V. Analysis

A. *Impact of slave nodes amount on CPU usage*

A deeper analysis reveals a number of characteristics of the multiplication of nodes on processing time and on resources overall. For instance, CPU usage in a 1+2 configuration is not optimal: one node remains at 100% while the other stops being used at around half of the processing.

B. *Impact of slave nodes amount on Disk usage*

Overall, we can observe indeed a change of IO behaviour when the amount of nodes changes:

- 1+2 configuration requires more frequent IO reads / writes, which can be explained by the limited amount of RAM available in this setup. Caching of smaller data parts, more frequent is therefore necessary.
- 1+4 configuration shows indeed less frequent IO operations. Their amount is however similar, and peak at a similar level as the 1+2 configuration.

C. *Impact of slave nodes amount on Network usage*

The 1+4 setup is more network intensive than the 1+2: most of the processing organisation seems to require access to data which is not located physically on a disk attached to the node where it is supposed to be processed. Instead of intense Disk IO in the 1+2 configuration, we therefore observe intense network IO in the 1+4 configuration. With higher amount of nodes in the cluster, we can observe an initial load on the network –most likely due to HDFS replication– which becomes less and less noticeable. Possibly this effect could be further attenuated if HDFS replication occurs in a more thorough manner.

D. *Impact of encryption: the 1+19 case*

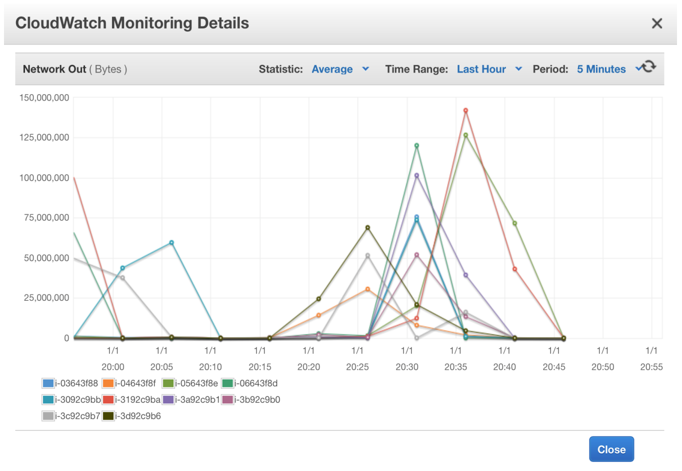
The 1+19 configuration is in our opinion the most interesting to analyse since it shows wholly how CPUs, Memory, and Disk IOs are effectively affected by the enabling of AES256 CTR.

1) *Observation 1*

When encryption is enabled, far more nodes reach 100% CPU usage over a part, or the entire period of processing. Without encryption, several nodes just end processing much before others. Encryption shows therefore a significant impact on CPU usage, which suggests that larger datasets processing may require either smart processing approaches (using data parts during decryption phases), or a horizontal upgrade of the cluster (i.e. more active nodes).

2) *Observation 2*

Our second observation relates to Network and Disk IO: we can see in both cases Read or Write spikes: we deduct that these are the moments where large parts of the data are transferred to RAM, and later when results are written back to HDFS. HDFS in such case could mean either a Disk Write and/or a Network Write, depending on which node is the data (source, and output) needs to be located, and whether it is replicated.


CloudWatch Monitoring Details

Network Out (Bytes) Statistic: Average Time Range: Last Hour Period: 5 Minutes

Fig. 15. Network Out (Bytes) behavior during 1+19 AES CTR processing (job start: 20h10)

Encryption is hardly noticeable on the measurements obtained (see Fig. 15).

- Disk reads spikes at 20.000.000 Bytes per 5min (exception made of the spike at around 82.000.000 Bytes per 5min on one node, probably due to data replication)
- Disk writes stagnate at around a median value of 150.000.000 Bytes, with spikes at 400.000.000 - 600.000.000 Bytes per 5min at the end of the processing (typically when results are written from RAM to persistent storage).
- Both Network reads and writes behaviour also seem unchanged by the use of encryption, and spike at around 100.000.000 - 120.000.000 Bytes per 5min.

If we then rule out the impact on nodes IOs, we are solely left with a noticeable impact on the nodes' use of CPU time, which confirms the literature description of how transparent encryption operates. Ultimately, multiplying the amount of nodes will help getting each data part fully loaded, decrypted, processed, and encrypted in RAM.

3) *Observation 3*

The first test using HDFS show a significant performance gain compared to the use of S3 buckets, around 3min gain in the 1+2 configuration. One possible explanation lies in the fact that the "HDFS volume" is "attached" to the EC2 instances, and directly accessible without having to suffer from network latency.

4) *Observation 4*

The performance on smaller datasets is not affected at all by encryption, in practice. This could be explained by the fact that encryption and decryption happens fully in RAM. This indeed implies that higher performance can be reached by adding more RAM to the cluster. During our experiment, the total amount of RAM at our disposal exceeded 218GB on the 1+19 nodes setup, which is much higher that the amount of data we wanted to process.

5) *Observation 5*

Encryption very seldom impacts the file size, which means that there is potentially no data overhead, and therefore no (or minimal) impact on Disks IO. Impact on network performance (set aside the use of RPC SASL) is also minimal for the very same reasons, and should remain similar to the performance levels of configurations without encryption used.

6) *Observation 6*

We can observe that the CPU resource is the most impacted by encryption, compared to other components. This verifies and supports the strategy of developing crypto-instructions within the CPU core. The impact of encryption (AES CTR) on newer CPUs could however remain significant, and CPU is the bottleneck to encryption, especially when random data searches are performed on the Big Data repository.

VI. Conclusions and Further Research

Encryption and decryption of course still represent a cost and impact performance; as popular belief rightly mentions, we observe a significant level of overhead due to encryption; it is however mitigated by hardware configuration, server software, ratio of dynamic versus static content, client distance to server, typical session length, and the caching strategy used both at server level or client level [15]. Real-life measurements gains presented in this research reveal that encryption forces the caching of larger sets of data; such decryption and encryption of larger datasets actually improve artificially the overall job performance. They also confirm that specific encryption modes (i.e. CTR encryption or decryption in forward mode) can indeed be performed in parallel [16]. Like any other processing payload, encryption is therefore massively distributed among nodes during processing.

In our tests, we have considered exclusively a linear-type (e.g. sequential) data processing. It occurs without "noticeable" performance impact, due to smart caching strategies and CPU-based encryption instructions sets. Random data access (e.g. using Hive) could however present a more significant challenge to performance. Future research should therefore focus on developing homomorphic encryption. It would greatly support the use of encrypted indexes, and limit the need for storing in RAM the data in the clear, leaving it potentially vulnerable to remote execution attacks.

Performance is however not the only potential impact of encryption: Key loss and Recovery / Proof of Retrievability (PoR) is a new control that needs to be considered as encryption at rest (or transparent encryption) is implemented at all. A crash of one KMS and the incapability of recovering encryption keys could be a disaster that leads to the loss of significant amounts of data. Likewise, keys theft remains a plausible attack scenario, and the protection of the KMS is of paramount importance, as well as the assurance of its high availability. This component could well be a more significant bottleneck than all the others.

Enabling encryption also requires that Centralised Authorisation is implemented together with Single Sign-on across all components: currently, authorisations mechanisms are per-component and unable for some to interact at all. There are therefore several authorisation steps required to access a single dataset.

Likewise, logging requires to be implemented consistently in all Hadoop components, and it requires to be protected adequately. The encryption of the logs produced by the various components (Hadoop core, HDFS) wasn't considered during this research. Many other Hadoop components also require protection and solutions are being researched (Hadoop core, HDFS, MapReduce, ZooKeeper, HBase, Hive, PIG, Oozie, Mahout, Flume, Sqoop) in Intel's driven 'Project Rhino' (GitHub, 2015).

Encryption no longer looks like an obstacle to performance and system usability. While precautions should always be taken (Key server high availability, Proof of Recovery), encryption can be seriously considered in any Big Data project.

We recommend that encryption zones (EZs) are made available by default in all HDFS distributions and that EZs are the default path used when uploading data. Where data needs to be randomly accessed, or where measured performance shows significant slowdown, one could choose to use the "unsecured" HDFS container instead.

REFERENCES

[1] NIST Big Data Public Working Group Security and Privacy Subgroup (2015) *NIST Big Data Interoperability Framework: Volume 4, Security and Privacy*. National Institute of Standards and Technology. doi: 10.6028/NIST.SP.1500-4.

[2] Fhom, H. S. (2015) 'Big Data: Opportunities and Privacy Challenges', arXiv.org, p. 823.

[3] Okman, L., Gal-Oz, N., Gonen, Y., Gudes, E. and Abramov, J. (2011) Security Issues in NoSQL Databases, 2011 IEEE 10th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom). IEEE, pp. 541–547. doi: 10.1109/TrustCom.2011.70.

[4] Quan, Q., Tian-Hong, W., Rui, Z. and Ming-jun, X. (2013) 'A model of cloud data secure storage based on HDFS'. IEEE, pp. 173–178. doi: 10.1109/ICIS.2013.6607836.

[5] Yin, J. and Zhao, D. (2015) 'Data confidentiality challenges in big data applications'. IEEE, pp. 2886–2888. doi: 10.1109/BigData.2015.7364111.

[6] Lafuente, G. (2015) 'The big data security challenge', Network Security, 2015(1), pp. 12–14. doi: 10.1016/S1353-4858(15)70009-7.

[7] Demchenko, Yuri, Peter Membrey, Cees de Laat, Defining Architecture Components of the Big Data Ecosystem. Second International Symposium on Big Data and Data Analytics in Collaboration (BDDAC 2014). Part of The 2014 Int. Conf. on Collaboration Technologies and Systems (CTS 2014), May 19-23, 2014, Minneapolis, USA.

[8] Cohen, J. and Acharya, S. (2013) 'Towards a Trusted Hadoop Storage Platform: Design Considerations of an AES Based Encryption Scheme with TPM Rooted Key Protections'. IEEE, pp. 444–451. doi: 10.1109/UIC-ATC.2013.57.

[9] Rong, C., Quan, Z. and Chakravorty, A. (2013) 'On Access Control Schemes for Hadoop Data Storage'. IEEE, pp. 641–645. doi: 10.1109/CLOUDCOM-ASIA.2013.82.

[10] Spivey, B. and Echeverria, J. in *Hadoop security*. Sebastopol, CA: O'Reilly Media, 2015

[11] Lakhe, B. in *Practical Hadoop Security*, 1st edition. Apress, 2014

[12] Huang, J., Nicol, D. M. and Campbell, R. H. (2014) 'Denial-of-Service Threat to Hadoop/YARN Clusters with Multi-tenancy'. IEEE, pp. 48–55. doi: 10.1109/BigData.Congress.2014.17.

[13] Rabl, T., Sadoghi, M., Jacobsen, H.-A., Gómez-Villamor, S., Muntés-Mulero, V. and Mankowskii, S. (2012) 'Solving Big Data Challenges for Enterprise Application Performance Management', arXiv.org.

[14] Lamb, C. (2015). Overview of HDFS Transparent Encryption. online Cloudera, Inc. Available at: http://www.slideshare.net/cloudera/hdfs-encryption Accessed 16 Jan. 2016.

[15] Desai, S., Park, Y., Gao, J., Chang, S.-Y. and Song, C. (2015) 'Improving Encryption Performance Using MapReduce'. IEEE, pp. 1350–1355. doi: 10.1109/HPCC-CSS-ICESS.2015.206.

[16] Dworkin, M. J. (2001) *Recommendation for block cipher modes of operation :*. 0 edn. Gaithersburg, MD: National Institute of Standards and Technology. doi: 10.6028/NIST.SP.800-38a.