Ανάλυση και Σχεδιασμός Πληροφοριακών Συστημάτων

Εξαμηνιαία Εργασία [Github Repository link : εδώ]

Time Series 3 : CrateDB - QuestDB

Εθνικό Μετσόβιο Πολυτεχνείο - Σχολή ΗΜΜΥ

Δημήτριος-Δαυίδ Γεροκωνσταντής
ΑΜ : *03119209*
dimitrisger2@gmail.com
el19209@mail.ntua.gr
ΣΗΜΜΥ ΕΜΠ

Αθανάσιος Τσουκλείδης-Καρυδάκης
ΑΜ : *03119009*
thanos.karidakis@gmail.com
el19009@mail.ntua.gr
ΣΗΜΜΥ ΕΜΠ

Φίλιππος Σεβαστάκης
ΑΜ : *03119183*
philipsevas@yahoo.gr
el19183@mail.ntua.gr
ΣΗΜΜΥ ΕΜΠ

***Abstract*—**Η παρούσα εργασία εκπονήθηκε στο πλαίσιο του μαθήματος "Ανάλυση και Σχεδιασμός Πληροφοριακών Συστημάτων" (Ροή Λ, Εξάμηνο 9ο) της Σχολής Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών του Εθνικού Μετσόβιου Πολυτεχνείου. Αντικείμενο της εργασίας αποτελεί η συγκριτική μελέτη της QuestDB και της CrateDB, δύο βάσεων δεδομένων προσανατολισμένων και προσαρμοσμένων για εφαρμογή σε time-series δεδομένα. Αφού παρουσιαστεί ο τρόπος εγκατάστασης του συστήματος που θα φιλοξενεί τις δύο βάσεις δεδομένων, ακολουθεί η μελέτη διαφόρων πτυχών της λειτουργικότητάς τους, με έμφαση στο performance που επιτυγχάνεται από αυτές. Μελετάται η επίδοση των δύο βάσεων δεδομένων στο loading δεδομένων (data ingestion) ποικίλων μεγεθών και υπό διαφορετικά configurations και στην παραγωγή και εκτέλεση ερωτημάτων (queries) διαφορετικών ειδών και απαιτήσεων ειδικά σε συνθήκες ύπαρξης μεγάλου όγκου δεδομένων. Κατά τη διάρκεια της παρούσας συγκριτικής μελέτης, αξιοποιούνται κατάλληλες μετρικές επίδοσης, ενώ τα αποτελέσματα της σύγκρισης επιχειρείται να αιτιολογηθούν βάσει των δομικών χαρακτηριστικών των δύο βάσεων. Τελικά, εξάγονται συμπεράσματα για τις συνθήκες υπό τις οποίες ενδείκνυται η χρήση καθεμιάς από τις δύο προς εξέταση βάσεις δεδομένων.

I. Σκοπός της Εργασίας

Σκοπός της παρούσας εργασίας αποτελεί η συγκριτική παρουσίαση και μελέτη δύο time-series databases, της CrateDB και της QuestDB. Με αφετηρία τους δεδομένους περιορισμούς και τα συγκεκριμένα δομικά/αρχιτεκτονικά χαρακτηριστικά τους, σκοπός είναι η ανάδειξη του τρόπου με τον οποίο αυτά μπορούν να επηρεάσουν το performance είτε κατά τη διάρκεια αποθήκευσης δεδομένων σε αυτές (στάδιο data loading/ingestion) είτε κατά τη διάρκεια υποβολής ερωτημάτων (queries) από χρήστες. Για την πλήρη μελέτη αυτών των πτυχών, σκοπός είναι το testing των δύο βάσεων δεδομένων υπό διαφορετικό φόρτο δεδομένων και με χρήση διάφορων configurations των δύο βάσεων αλλά και με ποικίλα και διαφορετικού είδους προς εκτέλεση ερώτηματα, ώστε να αναδειχθούν οι συνθήκες υπό τις οποίες μπορεί να επιτευχθεί το μέγιστο performance.

ΜΕΡΟΣ Α
Εγκαταστάση του Συστήματος Εισαγωγή σε Δομικά και Θεωρητικά Στοιχεία των δύο Βάσεων Δεδομένων

II. Εγκατάσταση του Συστήματος

A. Διαμόρφωση *cluster* μηχανημάτων

Το πρώτο βήμα στην εγκατάσταση ενός συστήματος που πρόκειται να φιλοξενήσει βάσεις δεδομένων (με την μια μάλιστα εκ των οποίων να απαιτεί πολλαπλά μηχανήματα ως κατανεμημένη) είναι η διαμόρφωση ενός cluster από υπολογιστές διασυνδεδεμένους μεταξύ τους. Πρόκειται να χρησιμοποιηθούν 3 Virtual Machines παρεχόμενα από την IAAS (Infrastructure-as-a-Service) υπηρεσία okeanos-knossos, στην οποία δημιουργήθηκε ακαδημαϊκός λογαριασμός. Στη συνέχεια, παρουσιάζονται με τη σειρα τα βήματα που ακολουθήθηκαν για το στήσιμο του cluster από remote machines.

1) Δημιουργία Ζεύγους Ιδιωτικού-Δημόσιου Κλειδιού
   Για την σύνδεση στα virtual machines του okeanos

θα πρέπει να προηγηθεί η επιβεβαίωση της αυθεντικοποιημένης πρόσβασης μέσω της χρήσης ενός ζεύγους κρυπτογραφικών κλειδιών. Στο μηχάνημα (π.χ. προσωπικό υπολογιστή) μέσω του οποίου θα αποκτούμε πρόσβαση στα VMs, γράφουμε:

```
$ ssh-keygen
```

Λαμβάνουμε στο terminal το δημιουργηθέν δημόσιο κλειδί με :

```
$ cat ~/.ssh/id_rsa
```

Καταχωρούμε το συγκεκριμένο δημόσιο κλειδί στην υπηρεσία του okeanos στο section "public keys". Την διαδικασία αυτή ακολούθησαν όλα τα μέλη της ομάδας (3) στους προσωπικούς τους υπολογιστές και έτσι καταχωρήθηκαν 3 δημόσια κλειδιά.

2) Δημιουργία Virtual Machines
Δημιουργούμε 3 μηχανήματα (section "machines") με 4 CPUs, 8GB RAM, 30GB disk space έκαστο. Στα μηχανήματα αυτά είναι εγκατεστημένο λειτουργικό σύστημα Ubuntu 16.04 LTS. Κατά τη δημιουργία, φροντίζουμε να κάνουμε select τα καταλληλα public keys που θα αναγνωρίζει το μηχάνημα. Πρόκειται για τα public keys που δημιουργήθηκαν προηγουμένως. Συνδεόμαστε στα μηχανήματά μας μέσω ssh :

```
$ ssh user@snf-****-ok-kno.grnetcloud.net
```

όπου "snf-****-ok-kno.grnetcloud.net" το εκάστοτε δημιουργηθέν μηχάνημα.

3) Διαμόρφωση Δικτυακής Εγκατάστασης
Το cluster θα είναι έτσι διαμορφωμένο ώστε το ένα μηχάνημα να έχει το ρόλο του coordinator (master) και τα υπόλοιπα το ρόλο των participants (workers). Απαιτείται η διαμόρφωση τόσο ενός public δικτύου μέσω του οποίου "ο εξώ κόσμος" θα μπορεί να επικοινωνεί με το cluster όσο και ενός private network μέσω του οποίου θα επικοινωνούν τα VMs μεταξύ τους. Για το public δίκτυο, διαθέτουμε μια public IP, την οποία και δημιουργούμε (okeanos section "IP addresses") και αναθέτουμε σε ένα από τα μηχανήματά μας, για ευνόητους λόγους στον master. Για ευκολία, μετανομάζουμε τους hosts (VMs) από το terminal τους ως εξής :

```
$ sudo hostnamectl set-hostname okeanos-
    ISmaster
$ sudo hostnamectl set-hostname okeanos-
    ISworker1
$ sudo hostnamectl set-hostname okeanos-
    ISworker2
```

Στο section "networks" δημιουργούμε ένα νέο private network στο οποίο προσθέτουμε τα 3 VMs. Εντός αυτού του private network, στους hosts έχουν αποδωθεί (ή μπορούμε explicitly να αποδώσουμε) IPv4 addresses της μορφής 192.168.1.2 (master), 192.168.1.3 (worker1),192.168.1.4 (worker2). Όπως είναι γνωστό, για την συνδεσιμότητα των μηχανημάτων με τον "έξω κόσμο" αυτές οι διευθύνσεις μεταφράζονται κατάλληλα (Network Address Translation - NAT).

Έπειτα, τροποποιούμε κατάλληλα το αρχείο /etc/hosts του κάθε host, γνωστοποιώντας σε όλους τους nodes, τους hosts του private network :

```
1 127.0.0.1       localhost
2 192.168.1.2     okeanos-ISmaster
3 192.168.1.3     okeanos-ISworker1
4 192.168.1.4     okeanos-ISworker2
5
6 # The following lines are desirable for IPv6
    capable hosts
7 ::1   localhost ip6-localhost ip6-loopback
8 ff02::1         ip6-allnodes
9 ff02::2         ip6-allrouters
```

Επανεκκινούμε τους hosts και προχωρούμε στη δημιουργία ζεύγους κρυπτογραφικών κλειδιών για την επικοινωνία μεταξύ των hosts. Στον master δημιουργούμε ζεύγος κλειδιών και μεταφέρουμε το δημιουργηθέν public key στον φάκελο authorized_keys. Εκτυπώνουμε το δημόσιο κλειδί και το κάνουμε copy:

```
$ ssh-keygen -t rsa
$ cat ~/.ssh/id_rsa.pub >> ~/.ssh/
    authorized_keys
$ cat ~/.ssh/id_rsa.pub
```

Στους δύο workers, κάνουμε επικόλληση του παραπάνω κλειδιού στον φάκελο authorized_keys :

```
$ vim ~/.ssh/authorized_keys
```

Με

```
ssh <hostname>
```

συνδεόμαστε από το ένα μηχάνημα στο μηχάνημα με όνομα hostname.

4) Αναβάθμιση Ubuntu
Αναβαθμίζουμε σταδιακά τα Ubuntu 16.04 LTS σε Ubuntu 20.04 LTS σε όλα τα (3) μηχανήματα. Για την μετάβαση από τα 16 στα 18, εκτελούμε :

```
$ sudo apt update && sudo apt upgrade -y
```

επιλέγοντας, όταν μας ζητηθεί, "Install the package maintainer's version" και αφού επανεκκινήσουμε το μηχάνημα, εκτελούμε :

```
$ sudo do-release-upgrade
```

επιλέγοντας πάλι όποτε μας ζητείται "Install the package maintainer's version" και επιπλέον επιλέγουμε το /dev/vda ως το GRUB install device. Από τα Ubuntu 18 στα 20, εκτελούμε πάλι

```
$ sudo do-release-upgrade
```

επιλέγοντας "yes" σε τυχόν ερωτήσεις του συστήματος και την 4.0 ως έκδοση αναβάθμισης του LXD snap.

Πιθανά Προβλήματα: Σε περίπτωση αδυναμίας αναβάθμισης του λειτουργικού συστήματος ή γενικώς σε περιπτώσεις στις οποίες φαίνεται ότι το σύστημα αντιμετωπίζει κάποια αδυναμία συνδεσιμότητας με τον έξω κόσμο, προτείνεται ο έλεγχος του routing table με :

```
$ netstat -nr
```

Εάν ως πρώτη διεύθυνση εμφανίζεται κάποια της μορφής 192.168.1.1 και όχι η default gateway του public δικτύου (83.212. ...), συνίσταται η διαγραφή της με :

```
$ sudo route delete default gw 192.168.1.1 eth1
```

B. Εγκατάσταση της *CrateDB*

Σκοπός μας είναι να τρέξουμε την CrateDB στο cluster. Για την εγκατάσταση της CrateDB σε κάθε κόμβο του cluster απαιτείται να έχουμε εγκαταστήσει μια εκ των παρακάτω εκδόσεων του Ubuntu σε κάθε κόμβο:

1) Ubuntu 20.04 LTS (Focal Fossa) - η επιλογή μας
2) Ubuntu 18.04.5 LTS (Bionic Beaver)
3) Ubuntu 16.04.7 LTS (Xenial Xerus)

Η CrateDB είναι κατανεμημένη βάση δεδομένων και απαιτεί multi-node setup. Ξεκινάμε εγκαθιστώντας την CrateDB σε κάθε κόμβο του συστήματος (με την μορφή ενός single-node installation) [3]. Αρχικά πρέπει το σύστημά μας να εντοπίσει και να κατεβάσει τα απαραίτητα πακέτα για την εγκατάσταση. Για την μετέπειτα εγκατάσταση θα χρησιμοποιήσουμε την εντολή apt που χρησιμοποιεί το Advanced Packaging Tool των Ubuntu για να κατεβάσουμε τα απαραίτητα πακέτα. Εκτελούμε τις ακόλουθες δύο εντολές:

```
$ sudo apt update
$ sudo apt install --yes apt-transport-https apt-
    utils curl gnupg lsb-release
```

Έπειτα κατεβάζουμε το public GPG key ώστε να μπορούμε να επιβεβαιώσουμε την γνησιότητα όσων πρόκειται να κατεβάσουμε στην συνέχεια.

```
$ curl -sS https://cdn.crate.io/downloads/deb/DEB-
    GPG-KEY-crate | sudo tee /etc/apt/trusted.gpg.d/
    cratedb.asc
```

Στη συνέχεια εντοπίζουμε το repository από το οποίο θα εγκαταστήσουμε την CrateDB και το προσθέτουμε στη λίστα από repositories του συστήματός μας του package manager apt. Έπειτα από αυτό, ο apt θα δύναται να έχει πρόσβαση στα πακέτα του repository της CrateDB:

```
[[ $(lsb_release --id --short) = "Debian" ]] &&
    repository="apt"
[[ $(lsb_release --id --short) = "Ubuntu" ]] &&
    repository="deb"
distribution=$(lsb_release --codename --short)

echo "deb [signed-by=/etc/apt/trusted.gpg.d/cratedb.
    asc arch=amd64] https://cdn.crate.io/downloads/$
    {repository}/stable/ ${distribution} main" \
| sudo tee /etc/apt/sources.list.d/cratedb.list
```

Με την εκτέλεση των ανωτέρω εντολών έχουμε μεταφέρει σε αρχεία του συστήματός μας (cratedb.list, cratedb.asc) τις απαραίτητες πληροφορίες ώστε να εγκαταστήσουμε την CrateDB μέσω του package manager apt. Προχωράμε στις ακόλουθες δύο εντολές για την εγκτάσταση της CrateDB:

```
$ sudo apt update
$ sudo apt install crate
```

Η ανωτέρω διαδικασία πρέπει να επαναληφθεί σε κάθε κόμβο του cluster. Στο σημείο αυτό έχουμε εγκαταστήσει σε κάθε κόμβο μια single-node installation της βάσης. Μπορούμε να ελέγξουμε τη βάση μας με τις ακόλουθες εντολές:

```
$ sudo systemctl start crate # start the database
$ sudo systemctl stop crate # stop the database
$ sudo systemctl restart crate # restart the DB
$ sudo systemctl status crate # check the status of
    the DB
```

Μπορούμε στη συνέχεια να τροποποιήσουμε και όσες μεταβλητές περιβάλλοντος επιθυμούμε στο path /etc/default-/crate. Ενδεικτικά, τροποποιήσαμε τα ακόλουθα:

```
CRATE_HEAP_SIZE=2g
MAX_LOCKED_MEMORY=unlimited
CRATE_USE_IPV4=true
```

Δεδομένου ότι επιθυμούμε να χρησιμοποιήσουμε και τους τρεις κόμβους του cluster ώστε να αξιοποιήσουμε την κατανεμημένη φύση της CrateDB, θα πρέπει να διαμορφώσουμε το απαραίτητο configuration της βάσης μας ώστε να πραγματοποιήσουμε ένα multi-node setup [4]. Στο αρχείο crate.yml του κάθε κόμβου που βρίσκεται στο directory /etc/crate προσθέτουμε τις ακόλουθες γραμμές κώδικα ώστε ο κάθε κόμβος να μπορεί να βρει και να συνδεθεί με όλους τους κόμβους του συστήματος. Η port 4300 είναι η default port για την επικοινωνία μεταξύ κόμβων σε ένα CrateDB cluster:

```
discovery.seed_hosts:
- 192.168.1.2:4300
- 192.168.1.3:4300
- 192.168.1.4:4300
```

Οι κόμβοι που αποτελούν το cluster απαιτείται να εκλέξουν έναν master κόμβο κατά το πρώτο bootstrapping του συστήματος. Ως εκ τούτου, πρέπει κάθε κόμβος να γνωρίζει ποιοι είναι οι κόμβοι που δύνανται να γίνουν master ώστε να προχωρήσει το σύστημα στην εκλογή του master. Αυτό δηλώνεται με τις ακόλουθες γραμμές κώδικα (θεωρούμε και τους τρεις κόμβους του cluster ως eligible να γίνουν master):

```
 cluster.initial_master_nodes:
- 192.168.1.2
- 192.168.1.3
- 192.168.1.4
```

Έπειτα πρέπει να ρυθμίσουμε τη βάση ώστε να γνωρίζει ότι αποτελείται από 3 κόμβους τους οποίους πρέπει να περιμένει να είναι ενεργοί προτού προχωρήσει σε recovery. Αυτό είναι απαραίτητο καθώς αν κάποιος κόμβος χαθεί κατά τη λειτουργία της βάσης για σύντομο χρονικό διάστημα, υπάρχει η περίπτωση η Crate να προβεί σε νέο balancing των δεδομένων στους εναπομείναντες μόνο κόμβους αλλάζοντας έτσι παραμέτρους του συστήματος όπως το sharding και δημιουργώντας ξανά replicas. Με την εκ νέου ενεργοποίηση του «νεκρού» κόμβου, η βάση θα προσαρμοστεί ξανά σε λειτουργία με 3 κόμβους, όμως κάτι τέτοιο είναι ακριβό και δεν συμφέρει αν οι κόμβοι που «πέφτουν» επανέρχονται σύντομα στο cluster. Οι ρυθμίσεις αυτές πραγματοποιούνται με τις ακόλουθες γραμμές κώδικα:

```
gateway:
        recover_after_data_nodes: 3
        expected_data_nodes: 3
```

Θα ρυθμίσουμε επίσης δύο σημαντικές παραμέτρους. Με το network.bind_host δηλώνουμε σε ποιες IP διευθύνσεις

θα κάνουμε bind την CrateDB. Θέτοντάς το στο 0.0.0.0, επιτρέπουμε στην CrateDB να ακούει όλα τα network interfaces για εισερχόμενες αιτήσεις. Επιπροσθέτως, με το network.publish_host, ο κάθε κόμβος δημοσιεύει στο υπόλοιπο cluster την διεύθυνση με την οποία κάποιος μπορεί να επικοινωνήσει μαζί του. Για τον κάθε κόμβο, θέτουμε το network.publish_host στην private IP address του:

```
network.bind_host: 0.0.0.0
network.publish_host: 192.168.1.2
# or 192.168.1.3 or 192.168.1.4
#(depends on the current node)
```

Τέλος ορίζουμε και το όνομα του cluster, αλλά και το όνομα του κάθε κόμβου του συστήματος:

```
cluster.name: my_cluster
node.name: <node-name>
```

όπου node-name = okeanos-ISmaster ή okeanos-ISworker1 ή okeanos-ISworker2 ανάλογα σε ποιον κόμβο βρισκόμαστε. Έχοντας πραγματοποιήσει το ανωτέρω configuration, η CrateDB τρέχει πλέον στο cluster τριών κόμβων. Για να την ενεργοποιήσουμε απαιτείται να εκτελέσουμε

```
$ sudo systemctl start crate
```

σε κάθε κόμβο του cluster. Στη συνέχεια, θα μπορούμε να αποκτήσουμε πρόσβαση στο Admin UI της CrateDB μέσω του http://localhost:4200/.
Σημείωση: Δεδομένου ότι ο χρήστης (εν προκειμένω εμείς) χρησιμοποιεί το τοπικό του μηχάνημα ενώ το cluster τρέχει στα VMs του okeanos, για να γίνει εφικτή η πρόσβαση στο Admin UI μέσω του localhost απαιτείται η χρήση SSH tunneling [6] κατά την σύνδεση του (από το τοπικό του μηχάνημα) σε έναν από τους remote κόμβους του CrateDB cluster (έστω στον master). Με αυτό τον τρόπο είναι δυνατή η ασφαλής κρυπτογραφημένη μεταφορά της κίνησης στην πόρτα 4200 (όπου τρέχει η CrateDB) του remote μηχανήματος στην αντίστοιχη πόρτα του τοπικού μηχανήματος. Με το όρισμα "-i" δίνεται το directory όπου είναι αποθηκευμένο το ιδιωτικό κλειδί που δημιουργήθηκε αρχικά, ώστε να μπορέσει να πραγματοποιηθεί με authorized τρόπο η σύνδεση:

```
ssh -i ~/.ssh/id_rsa -L 4200:localhost:4200 user@snf
    -39924.ok-kno.grnetcloud.net
```

C. Εγκατάσταση της *QuestDB*

Η QuestDB δεν είναι κατανεμημένη αλλά αντιθέτως απαιτεί single-node εγκατάσταση και άρα θα την εγκαταστήσουμε μόνο σε έναν κόμβο του cluster μας (και συγκεκριμένα στον master) [2]. Για την εγκατάστασή της θα χρησιμοποιήσουμε την εντολή wget ώστε να κατεβάσουμε το tar αρχείο (απευθείας από το διαδίκτυο μέσω HTTP/HTTPS) με τα απαραίτητα αρχεία, το οποίο βρίσκεται στο GitHub repository της QuestDB.

```
$ wget https://github.com/questdb/questdb/releases/
    download/7.3.7/questdb-7.3.7-rt-linux-amd64.tar.
    gz
```

Έπειτα θα κάνουμε extract το tar αρχείο που κατεβάσαμε:

```
$ tar -xvf questdb-7.3.7-rt-linux-amd64.tar.gz
```

Όντας στο directory ~/questdb-7.3.7-rt-linux-amd64/bin εκκινούμε την βάση:

```
$ ./questdb.sh start
```

Με χρήση του flag –d μπορούμε να επισημάνουμε και το root directory της QuestDB. Στο εξής, το default root directory για την QuestDB θα είναι το

```
$HOME/.questdb
```

εντός του οποίου βρίσκεται και ο φάκελος conf ο οποίος περιέχει το αρχείο server.conf που είναι το configuration file της QuestDB. Τέλος, με τις εντολές

```
$ ./questdb.sh stop
```

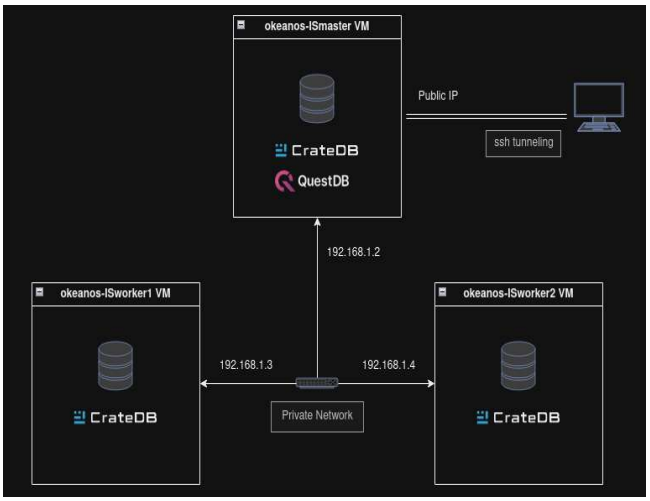
και

```
$ ./questdb.sh status
```

μπορούμε να σταματήσουμε και να δούμε την κατάσταση της βάσης αντίστοιχα. Έχοντας συνδεθεί κατάλληλα μέσω SSH tunneling (για λόγους που εξηγήθηκαν) στον master node με την ακόλουθη εντολή:

```
$ ssh -i ~/.ssh/id_rsa -L 9000:localhost:9000
    user@snf-39924.ok-kno.grnetcloud.net
```

μπορούμε να αποκτήσουμε πρόσβαση στο Admin UI της QuestDB μέσω του localhost: http://localhost:9000
Μετά την παραπάνω διαδικασία διαμόρφωσης του cluster και εγκατάστασης των βάσεων, το σύστημα που δημιουργείται απεικονίζεται σχηματικά στην εικόνα 1.



Εικόνα 1: Η δομή, τα components και η βασική δικτυακή σύνδεση του cluster

D. Εγκατάσταση του *TSBS*

Για την σύγκριση των δύο βάσεων θα χρησιμοποιήσουμε δεδομένα και queries τα οποία θα παραχθούν με την βοήθεια του TSBS (Time Series Benchmark Suite) [1]. Θα αξιοποιήσουμε συγκεκριμένα το use case Dev ops (μιας και είναι το μόνο συμβατό use case με την CrateDB) όπως θα αναλυθεί λεπτομερέστερα αργότερα στην αναφορά. Οι οδηγίες εγκατάστασης του TSBS παρατίθενται στο GitHub

repository των δημιουργών του στον ακόλουθο σύνδεσμο: https://github.com/timescale/tsbs. Προκειμένου να εγκατασταθεί επιτυχώς το TSBS απαιτείται να έχει εγκατασταθεί πρώτα η Go Programming Language [5]. Για να γίνει αυτό πρέπει αρχικά κατεβάσουμε με wget το απαιτούμενο συμπιεσμένο αρχείο ως εξής:

```
$ wget https://go.dev/dl/go1.21.5.linux-amd64.tar.gz
```

Έπειτα ελέγχουμε αν υπάρχει ήδη μια εγκατάσταση της Go στο σύστημά μας και αποσυμπιέζουμε το αρχείο κατάλληλα:

```
$ rm -rf /usr/local/go && tar -C /usr/local -xzf go1
    .21.5.linux-amd64.tar.gz
```

Στην συνέχεια προσθέτουμε την ακόλουθη γραμμή κώδικα στο αρχείο $HOME/.profile ώστε να τροποποιήσουμε την μεταβλητή περιβάλλοντος PATH:

```
export PATH=$PATH:/usr/local/go/bin
```

Εφαρμόζουμε τις αλλαγές εκτελώντας:

```
$ source $HOME/.profile
```

Επιβεβαιώνουμε ότι η Go εγκαταστάθηκε με επιτυχία εκτελώντας την εντολή:

```
$ go version
```

Σχόλιο: Προτού αρχίσουμε την εγκατάσταση του TSBS όπως αναφέρεται στον οδηγό, απαιτήθηκε να τρέξουμε την εντολή

```
$ go env -w GO111MODULE=off
```

Μόλις η εγκατάσταση του TSBS τελείωσε επιτυχώς, επαναφέραμε τα modules που απενεργοποιήθηκαν με την

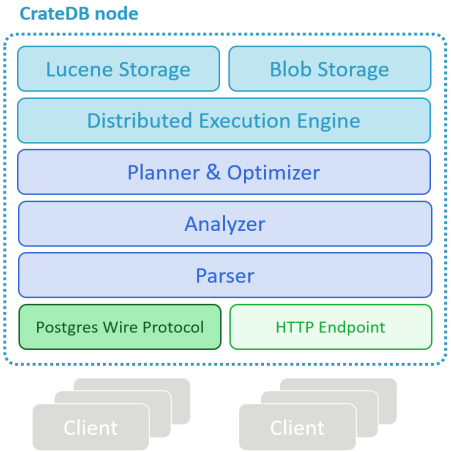
```
$ go env -w GO111MODULE=on
```

Από εδώ και στο εξής, είμαστε σε θέση να πραγματοποιήσουμε την σύγκριση μεταξύ των δύο βάσεων με βάση τα δεδομένα και τα queries που παράγονται με την βοήθεια του TSBS. Όντας εντός του directory ~/go/src/github.com/timescale/tsbs/cmd, μπορούμε να παράγουμε και να φορτώσουμε δεδομένα αλλά και να παράγουμε και να εκτελέσουμε queries στις δύο βάσεις. Οι εντολές που αξιοποιούνται για την κάθε διαδικασία παρατίθενται στην συνέχεια της αναφοράς παράλληλα με τις μετρήσεις που λάβαμε για διάφορες περιπτώσεις.

III. Η δομή και τα χαρακτηριστικά της CrateDB

A. Γενικά Χαρακτηριστικά

Η CrateDB [20] είναι μια κατανεμημένη time-series database που χρησιμοποιεί SQL interface ενώ ταυτόχρονα διαθέτει και noSQL χαρακτηριστικά (αναλαμβάνοντας τον χειρισμό μη σχεσιακών δεδομένων, όπως αρχείων JSON, documents, vectors, geospatial δεδομένων και binary αντικειμένων). Χαρακτηρίζεται από υψηλή κλιμακωσιμότητα (scalability) καθώς επιτυγχάνει πολύ ικανοποιητικές επιδόσεις κατά την προσθήκη περαιτέρω κόμβων στο cluster και από την αποδοτική και ταχεία εκτέλεση queries που στηρίζεται στο κατανεμημένο execution engine της, πράγμα θεμελιώδες κατά τη διαδικασία επεξεργασίας δεδομένων πραγματικού χρόνου. Δεδομένων των υψηλών απαιτήσεων των σύγχρονων data-driven εφαρμογών για γρήγορη εξυπηρέτηση αιτημάτων βασιζόμενων σε πραγματικού χρόνου και ετερογενή δεδομένα παραγόμενα από ποικίλες και διαφορετικών ειδών πηγές (sensors, databases, streams), η CrateDB προσφέρει μια ενοποιημένη αναπαράσταση πολλών ειδών δεδομένων, επιχειρώντας τον αποδοτικό χειρισμό τους μέσω μιας σύνθετης στοίβας συνιστώντων στοιχείων, όπως αυτή που φαίνεται στην εικόνα 2 [23].



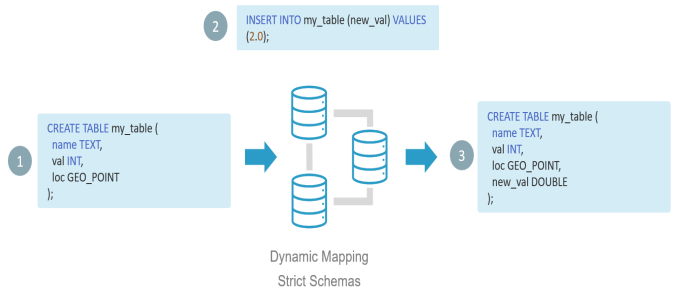
Εικόνα 2: Η στοίβα των στοιχείων που συναποτελούν έναν CrateDB κόμβο

Επιπλέον, η CrateDB βασίζεται σε shared-nothing αρχιτεκτονική [23], γεγονός που καθιστώντας όλους τους CrateDB nodes ισοδύναμους για να ικανοποιήσουν οποιοδήποτε αίτημα ενός client, εξασφαλίζει υψηλή διαθεσιμότητα (availability), isolation (η ανεξαρτησία των κόμβων δίνει την αίσθηση της απομονωμένης λειτουργίας του καθενός από αυτούς), scalability και εύκολα υλοποιήσιμο built-in-load-balancing των εισερχόμενων tasks στους CrateDB nodes. Βέβαια, το SQL interface της CrateDB την καθιστά εύκολη στη χρήση (programmability).

Η CrateDB χρησιμοποιεί columnar storage [22] διευκολύνοντας και κάνοντας πιο αποδοτικό το grouping των δεδομένων και τα aggregations επί αυτών, ενώ το query engine της είναι χτισμένο πάνω στο Apache Lucene Engine [21], μια βιβλιοθήκη για αποδοτικό text search [26] και indexing. Στην CrateDB όλα τα columns είναι by default indexed, ενώ οι αριθμητικές τιμές δεικτοδοτούνται με χρήση BKD-trees [22].

Η CrateDB, χάρη στην σύνθετη SQL-NoSQL αρχιτεκτονική της, χρησιμοποιεί Dynamic Schemas [24], με αποτέλεσμα να μπορεί να χειριστεί εύκολα και αποδοτικά όλα τα είδη δεδομένων (structured, semi-structured και unstructured). Κάτι τέτοιο είναι πολύ χρήσιμο στην ανάλυση χρονοεξαρτώμενων δεδομένων, όπου εμπλέκονται συνήθως πολλά διαφορετικά είδη δεδομένων (όπως JSON δεδομένα, Geospatial δεδομένα, blobs κ.λπ.). Για σχεσιακά δεδομένα, η CrateDB επιτρέπει την αλλαγή της δομής ενός πίνακα on the fly χωρίς την

ανάγκη εκ νέου ορισμού των πεδίων του όπως φαίνεται στην εικόνα 3 παρακάτω.



Εικόνα 3: Δυνατότητα on the fly αλλαγών στο structure της βάσης

Επιπρόσθετα, η CrateDB δύναται να αποθηκεύσει JSON αντικείμενα απευθείας, χωρίς ανάγκη για parsing του .json αρχείου και μετέπειτα ανάθεση του περιεχομένου των πεδίων του στις κατάλληλες στήλες (διαδικασία που αναμφίβολα επιβαρύνει τις επιδόσεις της βάσης). Η αποθήκευση ενός JSON αντικειμένου πραγματοποιείται με χρήση του τύπου δεδομένων OBJECT, ο οποίος μπορεί να περιέχει αυθαίρετο πλήθος από attributes και nesting levels (π.χ. πίνακες από αντικείμενα) και ανάλογα με τον τύπο του (DYNAMIC, STRICT, IGNORED) μπορεί να υποστηρίζει δυναμική αυτόματη ενημέρωση του σχήματος όταν προστίθενται νέα columns. Στην εικόνα 4 φαίνεται ο ορισμός ενός πίνακα που έχει ένα attribute τύπου OBJECT και ένα attribute τύπου πίνακα από OBJECTS.

```
>>> CREATE TABLE IF NOT EXISTS "doc"."sensors" (
"ts"    TIMESTAMP WITH TIME ZONE,
"type" TEXT,
"obj"   OBJECT(DYNAMIC) AS (
"MachineID" TEXT,
"Sensors" ARRAY(OBJECT(DYNAMIC)),
"Events" OBJECT(DYNAMIC)
)
    )
```

Εικόνα 4: Ο τύπος δεδομένων OBJECT

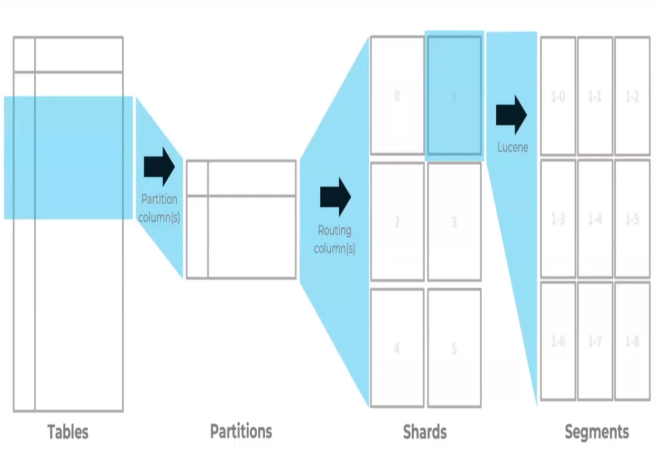
Τέλος, η κατανεμημένη φύση της CrateDB συνεπάγεται την απουσία spof (single point of failure), καθώς αν ένας κόμβος "κρασάρει" προσωρινά, ένας άλλος κόμβος μπορεί να αναλάβει τα δεδομένα του νεκρού κόμβου χάρη στην by default ύπαρξη αντιγράφων (replicas). Όταν ο νεκρός κόμβος επιδιορθωθεί και εισαχθεί ξανά στο cluster το σύστημα ισοσταθμίζει ξανά αυτόματα τα δεδομένα μεταξύ των κόμβων (automatic rebalancing) [25].

B. Replication

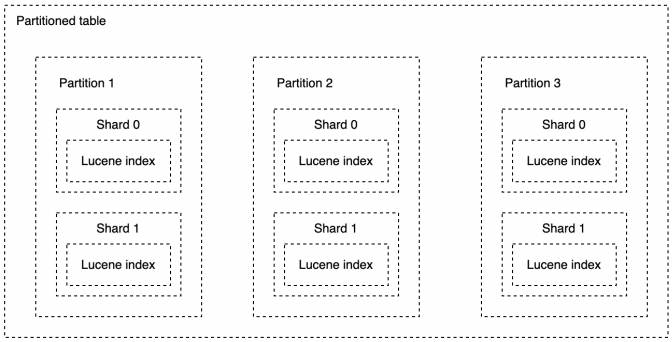
Σε ό,τι αφορά το replication [29], όταν αυτό είναι ενεργοποιημένο, εκτός του primary replica (αρχικού δεδομένου) διασκορπίζονται στο cluster secondary replicas, εξασφαλίζοντας το availability σε περίπτωση βλάβης κάποιου κόμβου, αλλά αυξάνοντας και το διαχειριστικό κόστος που περιλαμβάνει και την εξασφάλιση της συνέπειας μεταξύ των replicas. Το σύστημα της CrateDB -για λόγους επίδοσης- φροντίζει τα writes να γίνονται στο primary replica και σποραδικά να ενημερώνονται τα secondary replicas. Reads μπορεί να εξυπηρετήσει οποιοσδήποτε κόμβος, ακόμη κι αν διαθέτει secondary replica. Όταν ένας κόμβος πάθει κάποια βλάβη, το σύστημα εντοπίζει τα secondary replicas όλων των δεδομένων του κόμβου και για κάθε στοιχείο δεδομένων, αναβαθμίζει ένα secondary replica σε primary, ώστε αυτό το στοιχείο δεδομένων να εξακολουθεί να είναι διαθέσιμο. Η ενεργοποίηση του replication, λόγω του κόστους που ενέχει, προτιμάται όταν παρατηρούνται συχνές βλάβες των κόμβων, όταν για την τρέχουσα εφαρμογή η διαθεσιμότητα των δεδομένων αποτελεί προτεραιότητα και όταν η αξία των δεδομένων είναι σημαντική. Σε πειραματικές συνθήκες testing (όπως στην περίπτωσή μας) όπου τα δεδομένα ανακατασκευάζονται πολύ εύκολα και δεν παρατηρούνται συχνές βλάβες, το replication θα μπορούσε να χρησιμοποιηθεί μόνο για ερευνητικούς λόγους και δεν θα είχε κάποια πρακτική σημασία.

C. Sharding & Partitioning - Segments & Optimizations [28]

Η CrateDB προσφέρει τη δυνατότητα κάθε πίνακας να χωριστεί σε έναν αριθμό από shards [31] τα οποία είναι ισοκατανεμημένα στους κόμβους του cluster και στα οποία μοιράζεται όσο το δυνατόν ομοιόμορφα το σύνολο των γραμμών του εκάστοτε πίνακα της βάσης. Ταυτόχρονα, μέσω του partitioning [30], ο κάθε πίνακας μπορεί να μοιραστεί σε μικρότερα chunks με βάση τα values κάποιου attribute. Για παράδειγμα, ένας πίνακας μπορεί να χωριστεί σε υποπίνακες με βάση τον μήνα κατά τον οποίο καταγράφηκαν τα δεδομένα του (partition by month), δηλαδή -για παράδειγμα- να χωριστεί σε ένα chunk με δεδομένα που καταγράφηκαν τον μήνα Απρίλιο, σε ένα chunk με δεδομένα που καταγράφηκαν τον Μάιο κ.ο.κ. Συγκεκριμένα, κάθε πίνακας της CrateDB χωρίζεται σε partitions όπως περιγράφηκε βάσει των διαφορε-



Εικόνα 5: Shards - Partitions - Segments



Εικόνα 6: Η δομή και ο τρόπος αποθήκευσης ενός partitioned table

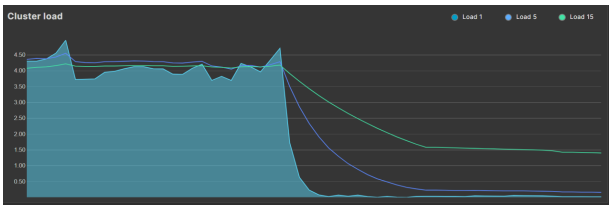
τικών τιμών ενός attribute, κάθε ένα από τα οποία χωρίζεται σε κάποιο αριθμό από shards (τα οποία ισοκατανέμονται στους CrateDB nodes) και τα οποία αποτελούνται από Lucene Segments πάνω στα οποία χτίζεται ένας Lucene index και τα οποία αποθηκεύονται φυσικά σε directory προσβάσιμο από τον κόμβο που διαχειρίζεται τα shards [εικόνες 5,6] [27]. Το sharding και το partitioning συμβάλλουν στον ορθό καταμερισμό τον δεδομένων μεταξύ των κόμβων ώστε να αποφευχθεί το underutilization κάποιων από αυτούς και στην γρηγορότερη εκτέλεση queries αφού πλέον οι πίνακες που πρέπει να διασχίσει το execution engine του κάθε κόμβου θα έχουν λιγότερες γραμμές (για παράδειγμα ένα query στοχευμένο στον μήνα Απρίλιο, θα χρειαστεί να αποκτήσει πρόσβαση -και μάλιστα αποδοτικά με indexes- μόνο στο αντίστοιχο partition). Είναι ενδιαφέρον να σημειωθεί ότι τα προαναφερθέντα Lucene segments είναι append-only. Κάθε φορά που γίνεται insert/delete/update σε δεδομένα, δημιουργούνται νέα segments και δεν μεταβάλλονται τα ήδη υπάρχοντα. Έτσι, διασφαλίζεται το immutability του δίσκου (και άρα το data recovery) και διευκολύνεται η διαχείριση των replicas των δεδομένων. Η append-only στρατηγική καθιστά πολύ γρηγορότερα τα writes αλλά με την πάροδο του χρόνου ενδέχεται να οδηγήσει σε υπέρμετρη αύξηση του πλήθους των segments, καθιστώντας αργά τα reads. Το πρόβλημα αυτό έρχεται να λύσει ο optimizer [32] της CrateDB που μεταξύ άλλων σποραδικά φροντίζει (αυτόματα) να συνενώνει (merge) μικρά segments σε μεγαλύτερα (φροντίζοντας για παράδειγμα να μην συμπεριλάβει segments που περιέχουν διαγεγραμμένα δεδομένα και κάνοντας έτσι ένα "καθάρισμα" των δεδομένων). Αυτός ο τρόπος optimization είναι configurable (μπορεί για παράδειγμα να ρυθμιστεί ο επιθυμητός μέγιστος αριθμός από segments που καταλήγουν να υπάρχουν στο τέλος του optimization). Σε ό,τι αφορά τον τρόπο ρύθμισης του sharding, του partitioning και των replicas, παρακάτω παρατίθεται ένα παράδειγμα ορισμού ενός πίνακα με 3 shards, partition με βάση τον μήνα και replication factor ίσο με 2 για κάθε πίνακα:

```
CREATE TABLE "metrics" (
........     //definition of attributes of the table
the_month TIMESTAMP GENERATED ALWAYS AS DATE_TRUNC('
    month', "timestamp"),
) CLUSTERED INTO 3 SHARDS
PARTITIONED BY (the_month)
WITH (number_of_replicas=2);
```

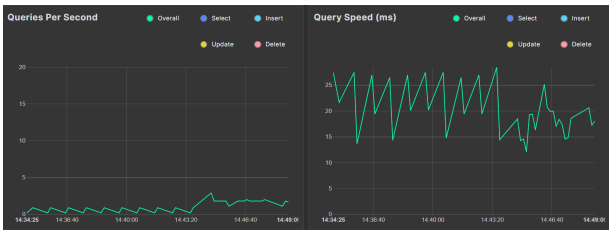
Τα δημιουργούμενα shards μοιράζονται δίκαια μεταξύ των κόμβων αυτόματα από την CrateDB, όπως αναφέρθηκε. Το κάθε shard συστήνεται να έχει μέγεθος μεταξύ 3GB και 70GB, ώστε να αποφεύγεται αφενός η δημιουργία πολλών μικρών shards, πράγμα που αυξάνει το διαχειριστικό overhead και αφετέρου η δημιουργία πολύ μεγάλων shards, πράγμα που δυσχεραίνει την επίδοση επιβαρύνοντας κάθε κόμβο με την ανάγκη διάσχισης πολύ μεγάλων πινάκων κατά την εκτέλεση ενός query. Επιπλέον, πρέπει να προσπαθούμε να χρησιμοποιούμε αριθμό shards που είναι πολλαπλάσιος του πλήθους των nodes του cluster μας ώστε να εξασφαλίζεται κατά το δυνατόν η ισοσταθμισμένη αποθήκευση δεδομένων στους κόμβους. Τέλος, το partitioning που χρησιμοποιούμε δεν πρέπει να μοιράζει τους πίνακες σε υπερβολικά πολλούς υποπίνακες, καθώς κάτι τέτοιο οδηγεί στην δημιουργία υπεράριθμων shards και άρα σε πολλαπλάσια αύξηση του διαχειριστικού overhead.

D. Admin UI

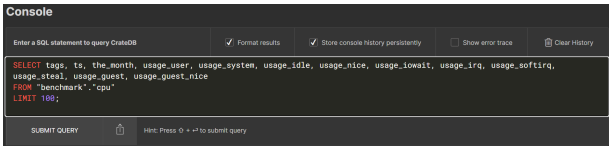
Το Admin UI της βάσης μπορεί να προσπελαστεί στη διεύθυνση http://localhost:4200/ και προσφέρει ενδιαφέροντα data visualizations αλλά και πληροφορίες για τους πίνακες της βάσης. Επίσης διαθέτει terminal για την εκτέλεση queries. Στην εικόνα 7 δεξιά παρατίθενται χαρακτηριστικά στιγμιότυπα από το Admin UI. Μάλιστα στην εικόνα 7e



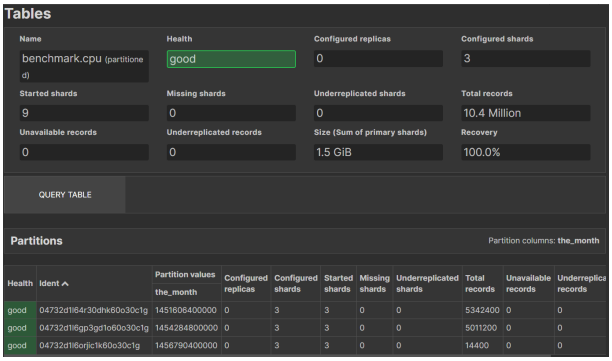
(a) Cluster Load



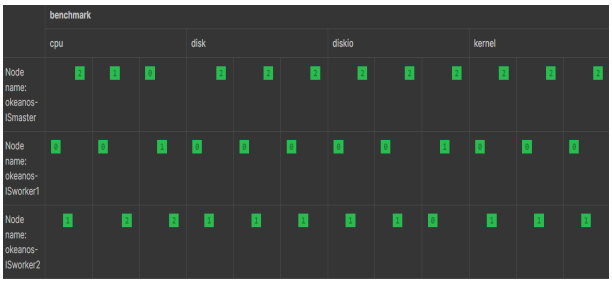
(b) Query Speed



(c) Console



(d) Tables



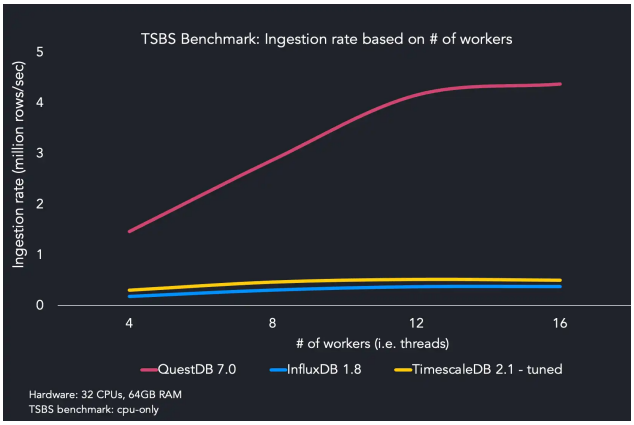
(e) Shards

Εικόνα 7: Στιγμιότυπα από το Admin UI της CrateDB

μπορούμε να δούμε την κατανομή των shards στους nodes. Μπορούμε να παρατηρήσουμε ότι στο στιγμιότυπο της εικόνας οι πίνακες (για παράδειγμα ο πίνακας cpu) έχουν χωριστεί σε 3 partitions, κάθε ένα από τα οποία έχει χωριστεί σε 3 shards. Κάθε κατακόρυφη στήλη (από τις συνολικά 3) του πίνακα cpu αφορά ένα partition. Μπορούμε για παράδειγμα να παρατηρήσουμε ότι το πρώτο partition (αριστερό) του πίνακα cpu αποτελείται από 3 shards (αριθμημένα ως 0, 1, 2), με το shard 0 να είναι αποθηκευμένο στον worker1, το shard 1 στον worker2 και το shard 2 στον master. Σε κάθε περίπτωση μπορούμε να παρατηρήσουμε ότι τα shards ισοκατανέμονται μεταξύ των nodes.

IV. Η δομή και τα χαρακτηριστικά της QuestDB

Η QuestDB είναι μια υψηλών επιδόσεων time-series database ικανή για τη διαχείριση μεγάλου όγκου δεδομένων. Έχει δοκιμαστεί σε μια ποικιλία σύγχρονων εφαρμογών που σχετίζονται με διαχείριση δεδομένων πραγματικού χρόνου, όπως ανάλυση δεδομένων αγοράς, δεδομένων Internet of Things και δεδομένων αισθητήρων. Όπως και η CrateDB, χαρακτηρίζεται από ευκολία χρήσης, χάρη στο SQL interface της. Η QuestDB, σύμφωνα με μετρήσεις από τους δημιουργούς της, υπερισχύει στον τομέα του data ingestion αισθητά σε σχέση με άλλες παρόμοιας κατηγορίας βάσεις δεδομένων όπως η InfluxDB, όπως φαίνεται στην εικόνα 8 όπου παρουσιάζεται συγκριτικά το ingestion rate με χρήση του benchmark cpu-only της σουίτας TSBS, η οποία χρησιμοποιείται αρκετά στην παρούσα εργασία [7].



Εικόνα 8: Ingestion Rate της QuestDB συγκριτικά με άλλες παρόμοιου είδους βάσεις δεδομένων

A. Data Insertion [9] - Deduplication [10] - WAL [11]

Βασικό πλεονέκτημα της QuestDB αποτελεί το υψηλό ingestion throughput της, δηλαδή η ικανότητα αποτελεσματικής αποθήκευσης των εισερχόμενων δεδομένων από κάποια εξωτερική πηγή, γεγονός υψίστης σημασίας σε εφαρμογές όπου εμπλέκονται δεδομένα πραγματικού χρόνου. Ειδικεύεται στην αποφυγή των ingestion bottlenecks επιλύοντας αποτελεσματικά ζητήματα όπως out-of-order data και duplicates [7]. Διατίθενται ποικίλοι τρόποι εισαγωγής δεδομένων στην Quest, καθένας από τους οποίους προτιμάται υπό διαφορετικές συνθήκες, όπως φαίνεται στην εικόνα 9. Ενδιαφέρον είναι να παρατηρηθεί ότι σε real time δεδομένα,

One-off data import

	CSV Upload	SQL COPY (Web Console)	InfluxDB Line Protocol	PostgreSQL
Sorted	✓	✓		
Lightly out of order	✓			
Heavily out of order		✓		

Periodic batch ingest

	CSV Upload	SQL COPY (Web Console)	InfluxDB Line Protocol	PostgreSQL
Sorted	✓		✓	✓
Lightly out of order	✓		✓	✓
Heavily out of order	✓		✓	✓

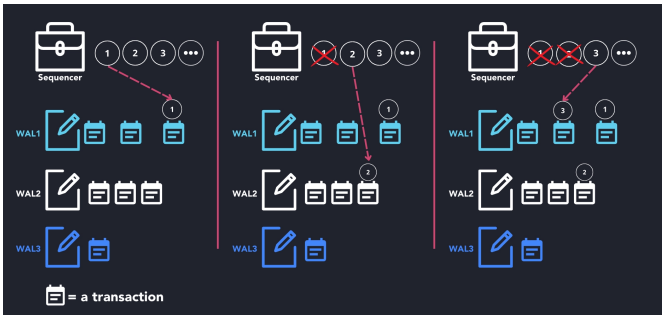
Real-time ingest

	CSV Upload	SQL COPY (Web Console)	InfluxDB Line Protocol	PostgreSQL
Sorted			✓	
Lightly out of order			✓	
Heavily out of order			✓	

Εικόνα 9: Ingestion methods για την QuestDB

ανεξαρτήτως του είδους τους (sorted, lightly/heavily out of order), η χρήση του InfluxDB Line Protocol για ingestion πρέπει να προτιμηθεί και μάλιστα μπορεί να επιτύχει εξαιρετικές επιδόσεις (στην περίπτωσή μας δεν προτιμάται αφού χρησιμοποιείται one-off data import). Σε κάθε περίπτωση, εκτός του παραπάνω τρόπου, υποστηρίζεται η

δυνατότητα για loading δεδομένων μέσω csv αρχείου, από το Web Console του UI και μέσω PostgreSQL [9]. Η QuestDB επιπλέον, διαθέτει μηχανισμό για αποδοτικό data deduplication. Σε περιπτώσεις όπου πιθανολογείται η ύπαρξη σφάλματος, ειδικά σε περιπτώσεις δεδομένων που αποθηκεύονται σταδικά σε πραγματικό χρόνο, η ενεργοποίηση του data deduplication επιτρέπει την επαναποστολή των δεδομένων στην βάση, χωρίς τον κίνδυνο δημιουργίας διπλότυπων δεδομένων. Έτσι, δεδομένα που είχαν καταφέρει να γραφτούν δεν θα αποθηκευτούν πάλι (idempotent table insert: πρόκειται για insert που δεν θα επηρεάσει ήδη υπάρχοντα rows), ενώ τα δεδομένα που είναι πράγματι καινούργια και δεν έχουν γραφτεί, θα προστεθούν [10]. Τέλος, η QuestDB επιτρέπει τη δυνατότητα λήψης δεδομένων από πάνω από μια πηγή (ingestion interface) ταυτόχρονα με χρήση των write-ahead-logs (WALs) καθένα από τα οποία λειτουργεί ως API για κάποιο συγκεκριμένο ingestion interface χωρίς να κλειδώνει (και άρα να χρησιμοποιεί αποκλειστικά) τον πίνακα στον οποίο προσπαθεί να προσθέσει δεδομένα. Για ένα table με ενεργοποιημένο WAL επιτρέπονται ταυτόχρονα DDLs και DMLs, κάτι που χωρίς WAL θα επιτρεπόταν μόνο μέσω του InfluxDB Line Protocol interface, η χρήση του οποίου όμως δεν είναι προτιμητέα πάντοτε. Τα transactions που υποβάλλονται μέσω των WALs των διαφόρων ingestion interfaces σειριοποιούνται με κάποιο transaction id από τον sequencer, συλλέγονται από την μονάδα WAL apply job και στέλνονται στον TableWriter που ενημερώνει την βάση και επιλύει τυχόν out of order δεδομένα [εικόνες 10, 11] [11].



Εικόνα 10: Ο Sequencer των WALs στην QuestDB

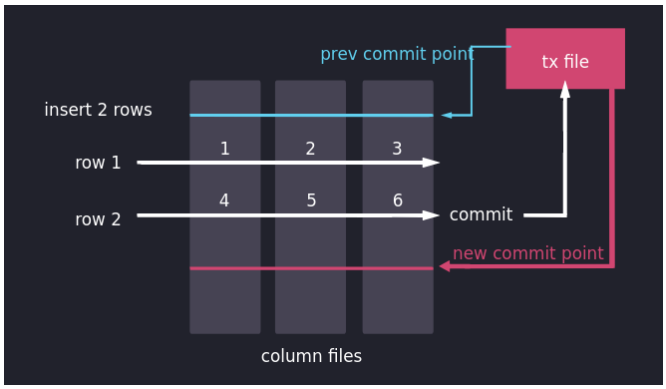
Εικόνα 11: Η λειτουργία των WALs στην QuestDB

B. Storage Model [12] της QuestDB

Η QuestDB χρησιμοποιεί columnar storage model (όπως και η CrateDB), γεγονός που συμβάλλει στην επίτευξη υψηλής επίδοσης κατά την εκτέλεση queries. Αυτό σημαίνει πως αποθηκεύει τα δεδομένα ανά στήλη αντί για ανά γραμμή, κάτι που είναι πολύ χρήσιμο στο τομέα των data analytics (OLAP) όπου τα queries περιέχουν πολύ συχνά aggregations (π.χ. max, sum, avg), τα οποία εφαρμόζονται σε μία μόνο στήλη και δεν απαιτούν γνώση των υπόλοιπων στηλών. Δεδομένου ότι πλέον τα στοιχεία ενός column αποτελούν συνεχόμενες θέσεις μνήμης, τα aggregations μπορούν να γίνουν αποδοτικότερα λόγω τοπικότητας των δεδομένων επί των οποίων εφαρμόζονται. Αντιθέτως, σε μια row oriented μνήμη για ένα aggregation, μαζί με το διάβασμα ενός στοιχείου κάποιου column, αντί να διαβάζονται με ένα read μιας cache line χρήσιμα στοιχεία του ίδιου column θα διαβάζονται μη χρήσιμα στοιχεία άλλων columns (ως συνεχόμενα στη μνήμη).

Στην QuestDB, η κάθε στήλη αποθηκεύεται ξεχωριστά σε κάποιο column file, το οποίο ακολουθεί -τόσο για τα insertions οσο και για τα updates- ένα append model (θυμίζοντας τα append only CrateDB segments). Οι στήλες των column files είναι randomly accessible. Ο αριθμός του row για το οποίο μας ενδιαφέρει η τιμή ενός column, μεταφράζεται κατάλληλα σε ένα offset εντός του αντίστοιχου memory page από όπου διαβάζεται η ζητούμενη τιμή.

Η QuestDB εξασφαλίζει isolation και consistency (κατά την ταυτόχρονη εκτέλεση πολλαπλών transactions) πραγματοποιώντας ατομικά τα table updates (atomic transaction updates), ενώ εξασφαλίζει συνέπεια σε περιπτώσεις query transactions που υποβάλλονται ταυτόχρονα με table updates ή insertions (επιλύοντας phantom φαινόμενα). Για να επιτύχει atomicity (all-or-nothing property), χρησιμοποιεί έναν counter (που μπορεί να αλλάζει ατομικά) ο οποίος καταγράφει το πλήθος των commited rows με τρόπο τέτοιο ώστε να αποτρέπει αναγνώσεις uncommited ή ενδιάμεσων δεδομένων που αντιπροσωπεύουν μια μη συνεπή κατάσταση της βάσης. Σε ό,τι αφορά το durability, η στατηγική βασίζεται στην εγγραφή των pages στον δίσκο όταν αυτά είναι dirty(αφήνεται δηλαδή ως ευθύνη του λειτουργικού συστήματος), παρέχοντας όμως τη δυνατότητα χειροκίνητης κλήσης kernel calls για εγγραφή column files στο δίσκο οποιαδήποτε στιγμή, με το κόστος όμως της μειωμένης απόδοσης. Στην εικόνα 12 φαίνεται ένα στιγμιότυπο του



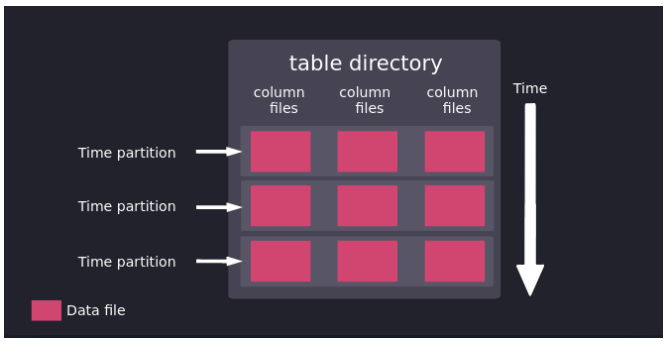
Εικόνα 12: Columnar Storage της QuestDB

storage της QuestDB, όπου ένα νέο transaction εισάγει δύο

rows κάνοντας τις κατάλληλες εγγραφές σε όλα τα column files (με append τρόπο) εκκινώντας από το προηγούμενο commit point (όπου τελείωσε ένα προηγούμενο transaction) που αποτελεί το tail του column file και καταλήγοντας σε ένα νέο commit και ενημέρωση του commit point (tail).
Σημειωτέον ότι το columnar storage model της QuestDB σε συνδυασμό με την δυνατότητά της για time-based partitioning (που αναφέρεται στη συνέχεια) αποτρέπουν την μείωση της επίδοσης για δεδομένα που χαρακτηρίζονται από υψηλό cardinality, δηλαδή από ύπαρξη πολλών μοναδικών τιμών ενός attribute, πράγμα που αποτελεί ένα αρκετά συχνό φαινόμενο σε time series databases όπου αρκετά attributes ενός πίνακα είναι tags (π.χ. Current CPU usage, μετρήσεις αισθητήρων κλπ) [18].

C. Partitions [13]

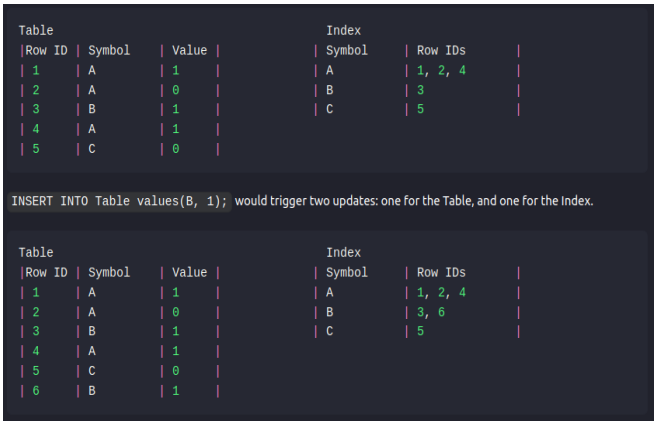
Όπως και στην CrateDB, ετσι και στην QuestDB επιτρέπεται η δημιουργία time-based partitions σε πίνακες με ορισμένο timestamp (designated timestamp). Τα column files διαχωρίζονται όπως στην εικόνα 13 σε time partitions, που είναι ικανά να μειώσουν το disk I/O overhead και το lookup time εστιάζοντας την αναζήτηση σε συγκεκριμένα partitions, διευκολύνουν τα interval scans (που παρουσιάζονται στη συνέχεια) και την διαχείριση των out of order commits μέσω του partition split που συμβάλλει στην μείωση του write amplification (ενός εγγενούς προβλήματος των flash memories/SSDs που οδηγεί σε αυξημένο όγκο φυσικά αποθηκευμένης πληροφορίας από αυτήν που λογικά προοριζόταν να αποθηκευτεί, συρρικνώνοντας τη διάρκεια ζωής και την απόδοση των memory devices) [17], [19]. Μέσω του partition split, όταν ένα νέο out of order row (π.χ. με timestamp μικρότερο από το ήδη υπάρχον μέγιστο) πρόκειται να προστεθεί, ένα υπάρχον partition (αυτό στο οποίο θα ανήκε το row) "σπάει" (στο σημείο όπου θα έπρεπε να προστεθεί με append τρόπο το νέο row) στο prefix sub-partition και στο suffix sub-partition, με το ένα εκ των οποίων να συμπεριλαμβάνει και το νέο out of order row. Σε διαφορετική περίπτωση, θα υπήρχε ο κίνδυνος το νέο row να σπαταλήσει άσκοπα όλο τον χώρο ενός νέου write page της μνήμης, καταλαμβάνοντας στην πραγματικότητα έναν πολύ μικρότερο χώρο αυτού και καταλήγοντας έτσι σε σπατάλη χώρου μνήμης. Τα παραπάνω subpartitions μπορούν στη συνέχεια να γίνουν squash, δηλαδή να ενοποιηθούν πάλι σε ένα μεγάλο partition που θα συμπεριλαμβάνει και το νέο row.



Εικόνα 13: Partitions στην QuestDB

D. Τύπος *Symbol* [14] και *Indexes* [15]

Η QuestDB υποστηρίζει έναν ενδιαφέροντα τύπο δεδομένων, τον Symbol. Αυτός αναφέρεται ουσιαστικά σε strings τα οποία όμως από το σύστημα της Quest αντιστοιχίζονται σε ακέραιους. Είναι χρήσιμος τύπος δεδομένων, ειδικά όταν για ένα πεδίο με strings, το cardinality του (δηλαδή το πλήθος των διαφορετικών strings που μπορεί να πάρει ως value) είναι μικρό, οπότε κάθε διαφορετικό string value μπορεί να αντιστοιχιστεί εύκολα σε έναν integer (π.χ. ένα string πεδίο "USA state" που παίρνει 50 δυνατές string τιμές, καθεμία εκ των οποίων συναντάται πιθανότατα σε πάρα πολλά rows μπορεί να αντιστοιχιστεί σε έναν integer). Πλέον αυτό το field αντιμετωπίζεται από την QuestDB (εσωτερικά και με διαφανή στον χρήστη τρόπο) ως ακέραιος, προσφέροντας ποικίλα πλεονεκτήματα (ευκολία διαχείρισης, εκτέλεσης πράξεων και συγκρίσεων αντί προσπέλαση και parsing του string και απλοποίηση του σχήματος της βάσης).
Τα columns τύπου symbol είναι τα μόνα που είναι indexed στην QuestDB προς το παρόν (σε αντίθεση με την Crate στην οποία δεικτοδοτούνται by default όλα τα columns). Ο index σε ένα symbol column, αποθηκεύει για κάθε value του symbol attribute τις γραμμές (row IDs) στις οποίες συναντάται αυτό το value, ώστε queries επί αυτού του column να εντοπίσουν τα επιθυμητά rows πολύ γρηγορότερα. Ωστόσο, η ύπαρξη του index ενέχει το κόστος της αποθήκευσης (storage space consumption) και διατήρησης/ενημέρωσής του. Πλέον κάθε insertion θα πρέπει εκτός του table να ενημερώνει και τον index, όπως φαίνεται στην εικόνα 14.

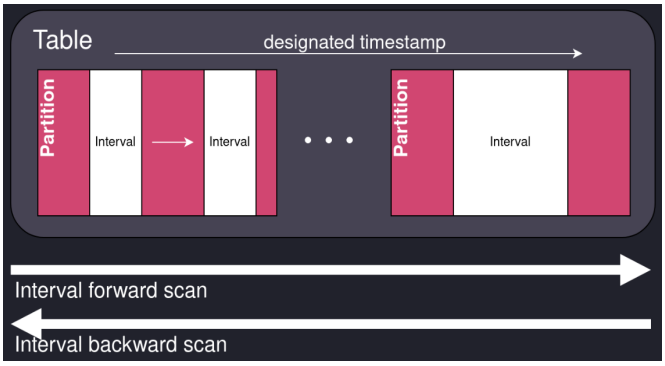


Εικόνα 14: Index στην QuestDB

E. Interval Scan [16]

Τέλος, παρουσιάζεται μια ενδιαφέρουσα λειτουργικότητα που υποστηρίζει η QuestDB, το Interval Scan, το οποίο βελτιώνει την επίδοση κατά την εκτέλεση queries με condition επί του ορισμένου timestamp. Σκοπός του interval

scan είναι ο αποδοτικός εντοπισμός των διαστημάτων (intervals) στα οποία αναφέρεται ένα time range query. Αρχικά η Quest εξετάζει το query και προσδιορίζει το condition επί του timestamp, εντοπίζει τα χρονικά διαστήματα (time intervals) που προσπελαύνει το query, βρίσκει τα άκρα αυτών των διαστημάτων με binary search (δυαδική αναζήτηση) στο ταξινομημένο ως προς το timestamp table (διαδικασία που προφανώς διευκολύνεται από την ύπαρξη time-based partitions) και τέλος κάνει scan (forward ή backward) μόνο στα intervals που οριοθετούνται από τα υπολογισθέντα boundaries [εικόνα 15]. Προφανώς, αυτή η διαδικασία απαιτεί (λόγω του binary search) τα δεδομένα να είναι οπωσδήποτε ταξινομημένα ως προς το χρόνο και για αυτό δεν εφαρμόζεται σε αποτελέσματα sub-queries τα οποία ενδέχεται να μην είναι σε timestamp order.



Εικόνα 15: Interval Scan στην QuestDB

ΜΕΡΟΣ Β

Μελέτη Επίδοσης των δύο Βάσεων Δεδομένων Data Ingestion και Query Execution Performance

V. Data Ingestion σε QuestDB και CrateDB

A. Μετρήσεις χρόνων και *memory consumption* κατά το *Data Ingestion*

Εκκινώντας την συγκριτική μελέτη μας σχετικά με την λειτουργικότητα και τις επιδόσεις των δύο βάσεων δεδομένων, εξετάζουμε το data ingestion. Πρόκειται να συγκρίνουμε τον απαιτούμενο χρόνο για το loading δεδομένων τριών διαφορετικών μεγεθών:

- small : 7776000 rows (δεδομένα 5 ημερών)
- medium : 48211200 rows (δεδομένα 1 μήνα)
- big : 93312000 rows (δεδομένα 2 μηνών)

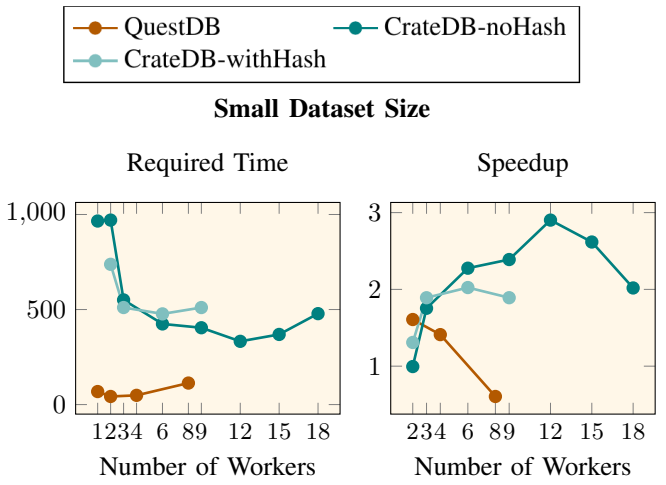
Σε αυτό το μέρος της ανάλυσης παράχθηκαν αυτόματα δεδομένα μέσω του TSBS (χρησιμοποιώντας το use case devops). Σε αυτό το σύνολο δεδομένων, μετρώνται ανά κάποιο time interval διάφορες μετρικές επίδοσης που επιτυγχάνουν διάφορα (9 στο πλήθος) components (cpu, memory, disk) κάποιου (configurable) αριθμού από μηχανήματα (hosts : παράμετρος scale). Επί παραδείγματι, για το data generation του μεγάλου dataset (που περιλαμβάνει δεδομένα 2 μηνών), εκτελούμε την παρακάτω εντολή στο directory ~/go/src/github.com/timescale/tsbs/cmd/tsbs_generate_data :

```
./tsbs_generate_data --use-case="devops" --seed=666\
--scale=20 \
--timestamp-start="2016-01-01T00:00:00Z" \
--timestamp-end="2016-03-01T00:00:00Z" \
--log-interval="10s" --format="questdb" \
| gzip > /tmp/questdbBIG-data.gz
```

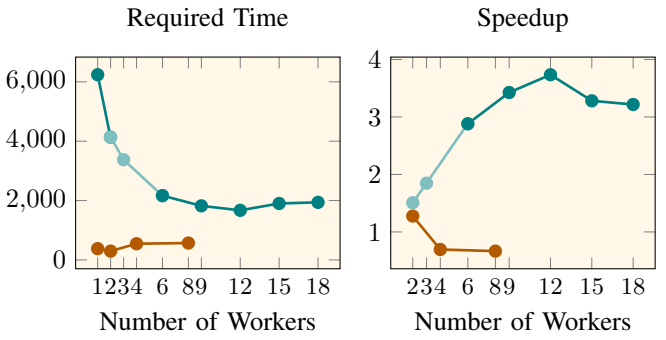
στην οποία προσαρμόζουμε κατάλληλα την βάση δεδομένων στην οποία αναφερόμαστε (format). Στη συνέχεια, για το loading των δεδομένων χρησιμοποιοείται αντίστοιχα η εντολή :

```
cat /tmp/questdbBIG-data.gz | gunzip | \
./tsbs_load_questdb \ --workers=1
```

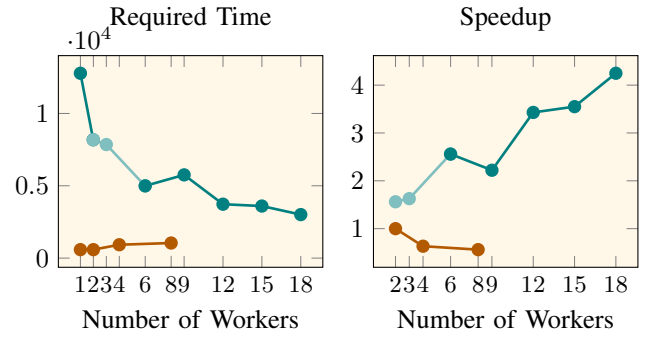
προσαρμόζοντας κατάλληλα το πλήθος των παράλληλων workers που θα εκτελέσουν το loading. Για το loading στην CrateDB, τα αρχεία main.go, creator.go μέσω των οποίων παράγεται το εκτελέσιμο ./tsbs_load_cratedb, αρχικά αφέθηκαν στην default εκδοχή τους, στην οποία δεν έχει προστεθεί partitioning, ενώ το πλήθος των shards είναι 5. Για να εξεταστεί καλύτερα η επίδραση αυτών των παραμέτρων στο data ingestion, στη συνέχεια θα αλλάξουμε αυτά τα αρχεία (όπως θα εξηγηθεί). Για την QuestDB έγινε loading των τριών datasets με 1, 2, 4 και 8 TSBS workers (οι οποίοι στέλνουν παράλληλα αιτήματα για insertion στην βάση), ενώ για την CrateDB χρησιμοποιήθηκαν έως και 18 TSBS workers και υπήρξε επίσης πειραματισμός ενεργοποιώντας την επιλογή hash-workers του TSBS, μέσω του οποίου γίνεται hashing των δεδομένων προς insertion ώστε το loading συγκεκριμένων συνόλων δεδομένων να το αναλαμβάνει πάντοτε ο ίδιος worker, προσφέροντας πλεονεκτήματα τοπικότητας [description : Whether to consistently hash insert data to the same workers] . Τα διαγράμματα που απεικονίζουν τα αποτελέσματα των μετρήσεων φαίνονται στην εικόνα 16. Η γενική εικόνα που λαμβάνουμε είναι ότι η επίδοση της QuestDB στο data ingestion είναι πολύ καλύτερη (σε ό,τι αφορά τον συνολικό απαιτούμενο χρόνο) σχετικά με την CrateDB. Αυτό είναι αναμενόμενο δεδομένου ότι η CrateDB επιφορτίζεται με τον διαχωρισμό των δεδομένων σε shards και στην κατανομή αυτών σε 3 διαφορετικούς κόμβους συνδεόμενους μέσω δικτύου. Σε ό,τι αφορά την χρήση της επιλογής των hash-workers, παρατηρούμε ότι αυτή μπορεί να βελτιώσει την επίδοση του data ingestion όταν χρησιμοποιούνται 2 ή 3 workers, ενώ για μεγαλύτερο αριθμό από workers η επίδοση γίνεται χειρότερη. Αυτό αρχικά οφείλεται στο overhead διαχείρισης του consistent hashing, το οποίο απαιτεί συνεχές hashing των δεδομένων και κατανομή τους σε workers. Όταν οι workers είναι περισσότεροι ενδεχομένως αυτή η διαδικασία μπορεί να γίνει επιβαρυντική στον απαιτούμενο χρόνο. Επίσης, ακόμη βασικότερο είναι το γεγονός ότι με χρήση πολλών workers, ο περιορισμός τους σε συγκεκριμένα σύνολα δεδομένων ενδέχεται να δρα περιοριστικά ως προς την παραλληλία οδηγώντας σε underutilization αυτών. Δεδομένων αυτών των παρατηρήσεων (που λήφθηκαν με βάση το μικρό dataset), για τα επόμενα datasets η επιλογή hash-workers ενεργοποιήθηκε μόνο με χρήση 2 και 3 workers και απενεργοποιήθηκε στις άλλες περιπτώσεις.



(a) Required Time per Number of Workers(b) Speedup per Number of Workers



(c) Required Time per Number of Workers(d) Speedup per Number of Workers



(e) Required Time per Number of Workers(f) Speedup per Number of Workers

Εικόνα 16: Απαιτούμενος χρόνος για data ingestion στις QuestDB και CrateDB ανά πλήθος χρησιμοποιούμενων workers και για διαφορετική ποσότητα δεδομένων (small-medium-big) και Speedup του data ingestion από τη χρήση πολλαπλών workers

Σε ό,τι αφορά την επίδραση της χρήσης πολλαπλών workers για το loading των δεδομένων, παρατηρούμε ότι στην QuestDB μικρή μόνο είναι η βελτίωση που παρατηρείται στο μικρό και μεσαίο dataset όταν χρησιμοποιούνται 2 workers. Με περισσότερους workers, η επίδοση επιδεινώνεται. Για την εξήγηση αυτού του φαινομένου σκεφτόμαστε ότι η QuestDB δουλεύει σε single-node mode και συγκεκριμένα σε ένα μόνο μηχάνημα με 4 CPUs οι οποίες βέβαια μοιράζονται την μνήμη. Αν θεωρήσουμε ότι τα αιτήματα των TSBS workers για την ικανοποίησή τους κατανέμονται ομοιόμορφα στις CPUs των κόμβων, οι 8 workers (δηλαδή 2 παράλληλα αιτήματα ανά CPU) υπερβαίνουν τις δυνατότητες παραλληλισμού του κόμβου ενώ ακόμη και η τακτική ενεργοποίησης πολυνηματισμού εντός των CPUs (που ενδεχομένως ακολουθεί και η QuestDB) οδηγεί σε contention στην κοινή RAM του μηχανήματος. Σχετικά με το Speedup που επιτυγχάνεται, αυτό είναι ικανοποιητικό μόνο για τους 2 workers (περίπου 1.7 - κοντά στο 2) και το μικρό dataset. Για την CrateDB, κατά το loading των small και medium datasets, ιδανική είναι η χρήση 12 workers (χωρίς βέβαια να επιτυγχάνεται ικανοποιητικό speedup : περίπου 3 με 4 <<12). Με ακόμη περισσότερους workers η επίδοση χειροτερεύει για το μικρό dataset (λίγα δεδομένα για επίτευξη μεγάλου παραλληλισμού), παραμένει σχεδόν σταθερή για το μεσαίο και βελτιώνεται λίγο για το μεγάλο dataset (του οποίου το μέγεθος ευνοεί μεγαλύτερο παραλληλισμό). Για λόγους προφανείς, ο απαιτούμενος χρόνος είναι ανάλογος του μεγέθους του dataset για συγκεκριμένο αριθμό από workers (π.χ. με 1 worker για το loading διπλάσιας ποσότητας -σε rows- δεδομένων απαιτείται διπλάσιος χρόνος).

Στη συνέχεια προχωρούμε στην μέτρηση επιδόσεων (στο μεγάλο πλέον dataset) κατά το data ingestion με χρήση διαφορετικού configuration της CrateDB, δηλαδή με χρήση (και μη) μηνιαίου partitioning (έχουμε δεδομένα 2 μηνών) και για διαφορετικό πλήθος από shards (ανά partition). Για να δηλώσουμε στο TSBS το configuration για το οποίο πραγματοποιούμε μετρήσεις πρέπει να τροποποιήσουμε τα αρχεία creator.go και main.go (πριν την φόρτωση των δεδομένων στη βάση) τα οποία βρίσκονται στο directory ~/go/src/github.com/timescale/tsbs/cmd/tsbs_load_cratedb. Φροντίζουμε να κάνουμε go build μετά τις τροποποιήσεις. Οι τροποποιήσεις που πρέπει να γίνουν (ενδεικτικά για χρήση configuration με 3 shards και με partitioning ανά μήνα) φαίνονται ακολούθως:

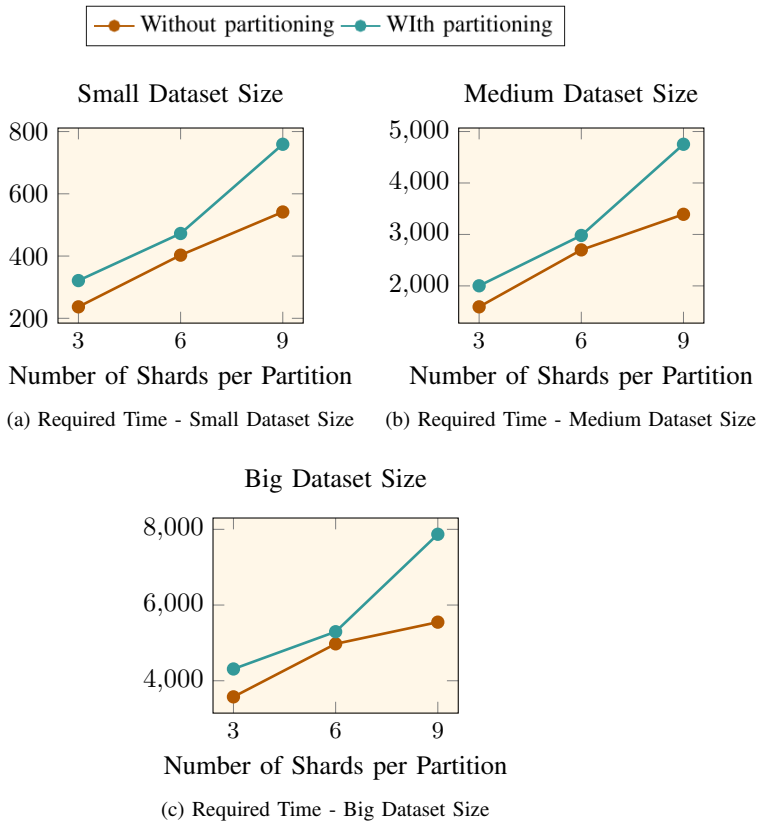
a) creator.go: Οι αλλαγές για το creator.go

```
CREATE TABLE %s (
tags object as (%s),
ts timestamp,
the_month TIMESTAMP GENERATED ALWAYS AS DATE_TRUNC('
    month', "ts"),
%s
) CLUSTERED INTO %d SHARDS
PARTITIONED BY (the_month)
WITH (number_of_replicas = %d)
```

b) main.go: Οι αλλαγές για το main.go

```
numShards := flag.Int("shards", 3, "Number of shards
    per a metric table")
```

Οι μετρήσεις απεικονίζονται στα διαγράμματα της εικόνας 17. Αρχικά, παρατηρούμε ότι με χρήση partition ο απαιτούμενος χρόνος είναι μεγαλύτερος καθώς προστίθεται σε αυτόν ο χρόνος που χρειάζεται για την δημιουργία των partitions. Επιπλέον, η ύπαρξη partitions αυτομάτως αυξάνει τα δημιουργούμενα shards (καθώς το πλήθος τους -π.χ. 3- υπολογίζεται ανά partition). Ομοίως, η αύξηση του πλήθους των shards οδηγεί σε αύξηση του απαιτούμενου χρόνου για data ingestion η οποία όμως δεν ακολουθεί γραμμική σχέση (για παράδειμα ο απαιτούμενος χρόνος με 9 shards είναι λιγότερος από τον τριπλάσιο του απαιτούμενου χρόνου για 3 shards).



(a) Required Time - Small Dataset Size

(b) Required Time - Medium Dataset Size

(c) Required Time - Big Dataset Size

Εικόνα 17: Απαιτούμενος χρόνος για data ingestion στην CrateDB με 12 workers για διαφορετικό πλήθος από shards και για διαφορετική ποσόσητα δεδομένων (small-medium-big).

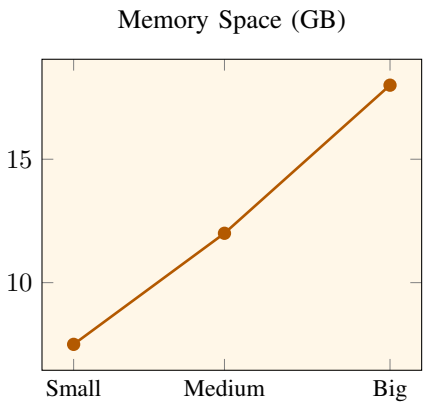
Τέλος, παρουσιάζεται ο τρόπος με τον οποίο το loading διαφορετικού μεγέθους δεδομένων επηρεάζει το memory space που απαιτείται για την αποθήκευσή τους. Για την εύρεση του memory space που καταναλώνεται, στην QuestDB χρησιμοποιούμε την εντολή :

```
$ du -hs ~/.questdb/db
```

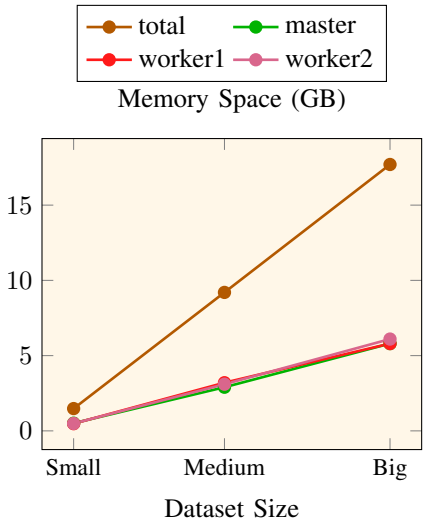
και για την CrateDB (και σε κάθε κόμβο ξεχωριστά) την εντολή :

```
$ du -hs /var/lib/crate
```

Τα αποτελέσματα φαίνονται στην εικόνα 18. Παρατηρούμε



(a) QuestDB



(b) CrateDB

Εικόνα 18: Memory Space Consumption ανά Dataset Size μετά το data ingestion σε QuestDB και CrateDB (Small: 7776000 rows, Medium: 48211200 rows, Big: 93312000 rows)

ότι στην CrateDB και για κάθε σύνολο δεδομένων (εκτός από το μεγάλο, όπου οη διαφορά είναι μικρή) απαιτείται λιγότερος χώρος μνήμης για την αποθήκευσή τους. Εκ πρώτης όψεως, το compression της CrateDB φαίνεται αποδοτικότερο συγκριτικά με αυτό της QuestDB. Το compression της Quest φαίνεται αποδοτικότερο όσο αυξάνεται το μέγεθος του συνόλου δεδομένων. Ενώ για το μικρό dataset, καταλαμβάνει σχετικά σημαντικό χώρο (7.5 GB), αυξανομένου του μεγέθους του dataset δεν καταλαμβάνει ανάλογα μεγάλο χώρο (για 7πλασιασμό των rows από το μικρό στο μεσαίο dataset, το memory consumption δεν καταλήγει ούτε καν να διπλασιαστεί). Ενδεχομένως, η QuestDB ενεργοποιεί το compression όταν το μέγεθος του συνόλου δεδομένων ξεπερνά κάποιο threshold. Αντιθέτως, η συμπεριφορά της CrateDB για κάθε σύνολο δεδομένων φαίνεται να είναι παρόμοια. Καταλαμβάνει ήδη μικρό χώρο μνήμης για το μικρό

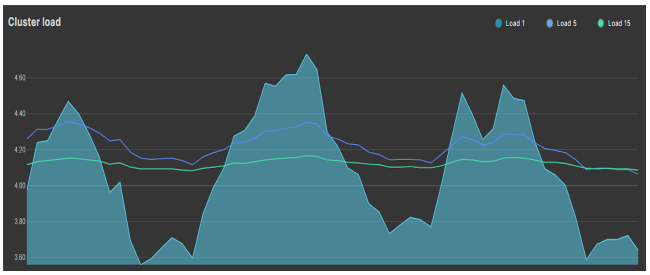
σύνολο δεδομένων (1.5GB), με 7πλασιασμό (σχεδόν) των rows σχεδόν 7πλασιάζεται η κατανάλωση χώρου (×6.3) ενώ με μετέπειτα 2πλασιασμό (σχεδόν) των rows 2πλασιάζει τον απαιτούμενο χώρο μνήμης (από 9.2 σε 17.7GB). Έτσι για το μεγάλο dataset, οι δύο βάσεις δεδομένων καταλήγουν να καταλαμβάνουν τον ίδιο σχεδόν συνολικό χώρο. Τέλος, να σημειωθεί ότι (όπως φαίνεται στην εικόνα 18b) η CrateDB ισοκατανέμει τα δεδομένα μεταξύ των τριών κόμβων ώστε σε καθέναν από αυτούς το memory consumption να είναι σχεδόν ίδιο.

B. Μετρικές χρήσης πόρων συστήματος κατά το *Data Ingestion*

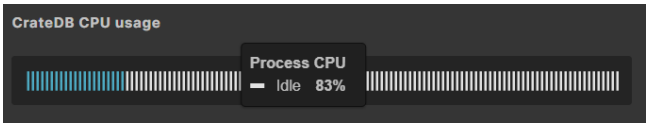
Κλείνοντας την μελέτη επί του Data Ingestion, παρουσιάζονται μετρήσεις από διάφορες μετρικές επίδοσης όπως αυτές δόθηκαν από το admin UI της CrateDB, όσο παραγμοτοποιούταν το loading του μεγάλου dataset. Σε ό,τι αφορά την QuestDB, το UI της δεν παρέχει τέτοιου είδους πληροφορίες. Θα ασχοληθούμε με τρεις μετρικές επίδοσης :

1) Cluster Load : Το ποσοστό του συνολικού capacity του εκάστοτε node που χρησιμοποιείται κατά την τρέχουσα χρονική στιγμή. Τιμές μεγαλύτερες του 1 (100%) υποδηλώνουν υπερχρησιμοποίηση των πόρων σε βαθμό που υπερβαίνει τις δυνατότητες των nodes. Κάτι τέτοιο οδηγεί σε αναμονή αιτημάτων που δεν μπορούν να ικανοποιηθούν την τρέχουσα χρονική στιγμή λόγω φόρτου.
2) CrateDB cpu usage : Ο βαθμός χρησιμοποίησης της CPU του εκάστοτε node από λειτουργίες της CrateDB
3) Heap usage : Ο βαθμός χρησιμοποίησης του capacity του heap του εκάστοτε κόμβου το οποίο χρησιμοποιείται για dynamic memory allocation.

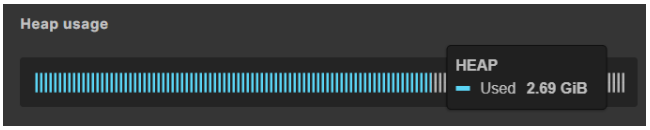
Στη συνέχεια, παρουσιάζονται στην εικόνα 19 οι παραπάνω μετρικές, όπως μετρήθηκαν με configuration τριών shards χωρίς partitioning (αν και σε κάθε configuration οι παρατηρήσεις ήταν πολύ παρόμοιες). Παρατηρούμε ότι το cluster load σε όλους τους κόμβους κυμαίνεται μεταξύ του 3.5 (350%) και 4.5 (450%) κατά τη διάρκεια της φόρτωσης δεδομένων, κάτι που σημαίνει ότι τα αιτήματα για insertion υπερφορτώνουν το σύστημα, όπως εξηγήθηκε παραπάνω. Μάλιστα, παρατηρήθηκαν σημαντικές διακυμάνσεις στο μέρος του cluster load που αφορά τον master node (load1 στο διάγραμμα) σε αντίθεση με τους άλλους δύο κόμβους οι οποίοι είχαν σχετικά σταθερό φόρτο. Σε ο,τι αφορά το cpu usage, αυτό σε όλους κόμβους έφτανε μέχρι περίπου το 27% (όπως φαίνεται στην εικόνα 19b) ενώ συχνά μειωνόταν έως το 24% και υπήρχαν περιπτώσεις παροδικής μείωσής του στα 6-7%. Συνεπώς, το cpu usage κατά το loading των δεδομένων δεν φαίνεται να είναι πολύ μεγάλο (σε σύγκριση μάλιστα με το αντίστοιχο usage κατά την εκτέλεση ορισμένων πιο απαιτητικών queries όπως θα παρουσιαστεί παρακάτω). Το heap usage είναι ωστόσο αρκετά υψηλό, φθάνοντας μέχρι και τα 3GB (από το συνολικό μέγεθος των 4GB) με παροδικές μειώσεις μέχρι και τα 1.2GB, ενώ το disk usage (που δεν παρατίθεται στην εικόνα) είναι επίσης αρκετά υψηλό, δεδομένου ότι το ingestion αφορά writes στον δίσκο. Το παραπάνω φόρτο στις συνιστώσες μνήμης του συστήματος (heap, disk) αλλά και στο cluster load εξηγείται από το γεγονός ότι γίνεται loading ενός αρκετά μεγάλου dataset.



(a) Cluster Load



(b) CPU Usage



(c) Heap Usage

Εικόνα 19: Μετρικές συστήματος κατά το Data Ingestion στην CrateDB

VI. Εκτέλεση QUERIES στις δύο βάσεις

A. Χρήση *batches* από *TSBS queries*

Στο μέρος αυτό, εξετάζεται η επίδοση των δύο βάσεων δεδομένων κατά την εκτέλεση batches από queries τα οποία έγιναν generate αυτόματα με χρήση του TSBS (με σκοπό την αξιολόγηση των δύο βάσεων σε πραγματικές συνθήκες κατά τις οποίες δέχονται πολλαπλά αιτήματα από διάφορες πηγές). Για την παραγωγή των queries με χρήση του TSBS θα χρειαστεί όντας στο directory ~/go/src/github.com/timescale/tsbs/cmd/tsbs_generate_queries να εκτελέσουμε μια εντολή της μορφής :

```
./tsbs_generate_queries --use-case="devops" \
--seed=666 --scale=20 \
--timestamp-start="2016-01-01T00:00:00Z"\
--timestamp-end="2016-03-01T00:00:01Z" \
--queries=100 --query-type="double-groupby-5" \
--format="cratedb" \
|gzip > /tmp/cratedb100-queries-double-groupby-5.gz
```

προσαρμόζοντας τον τύπο και το πλήθος των queries ("query-type" & "queries") και την βάση για την οποία θα δημιουργηθούν ("format"). Για λόγους πληρότητας, θα εκτελεστούν τα παρακάτω 4 είδη queries, των οποίων η λειτουργία και η υλοποίηση ανά βάση παρατίθενται παρακάτω:

1) single-groupby-1-1-1
   Περιγραφή : Υπολογισμός μέγιστης τιμής ανά λεπτό κάποιας μετρικής (cpu "usage_user") ενός host για τη

διάρκεια μιας ώρας
Υλοποίηση:

- CrateDB

```
SELECT
    date_trunc('minute', ts) as minute,
    max(usage_user) AS max_usage_user
FROM
    cpu
WHERE
     tags['hostname'] IN ('host_1')
    AND ts >= 1453521384428
    AND ts < 1453524984428
GROUP BY minute
ORDER BY minute ASC
```

- QuestDB

```
SELECT
    timestamp,
    max(usage_user) AS max_usage_user
FROM
    cpu
WHERE
    hostname IN ('host_1')
    AND timestamp >= '2016-01-23T03:56:24Z'
    AND timestamp < '2016-01-23T04:56:24Z'
SAMPLE BY 1m
```

2) single-groupby-5-1-12
Περιγραφή : Υπολογισμός μέγιστης τιμής ανά λεπτό 5 μετρικών της cpu ενός host για τη διάρκεια 12 ωρών
Υλοποίηση:

- CrateDB

```
SELECT
    date_trunc('minute', ts) as minute,
    max(usage_user) AS max_usage_user,
    max(usage_system) AS max_usage_system,
    max(usage_idle) AS max_usage_idle,
    max(usage_nice) AS max_usage_nice,
    max(usage_iowait) AS max_usage_iowait
FROM
    cpu
WHERE
    tags['hostname'] IN ('host_1')
    AND ts >= 1453028181428
    AND ts < 1453071381428
GROUP BY minute
ORDER BY minute ASC
```

- QuestDB

```
SELECT
    timestamp,
    max(usage_user) AS max_usage_user,
    max(usage_system) AS max_usage_system,
    max(usage_idle) AS max_usage_idle,
    max(usage_nice) AS max_usage_nice,
    max(usage_iowait) AS max_usage_iowait
FROM
    cpu
WHERE
    hostname IN ('host_1')
    AND timestamp >= '2016-01-17T10:56:21Z'
    AND timestamp < '2016-01-17T22:56:21Z'
SAMPLE BY 1m
```

3) double-groupby-5
Περιγραφή : Υπολογισμός για κάθε host των μέσων τιμών 5 μετρικών ανά ώρα και για συνολική διάρκεια 12 ωρών
Υλοποίηση:

- CrateDB

```
SELECT
    date_trunc('hour', ts) AS hour,
    mean(usage_user) AS mean_usage_user,
    mean(usage_system) AS mean_usage_system
        ,
    mean(usage_idle) AS mean_usage_idle,
    mean(usage_nice) AS mean_usage_nice,
    mean(usage_iowait) AS mean_usage_iowait
FROM
    cpu
WHERE
    ts >= 1455759268856
    AND ts < 1455802468856
GROUP BY hour, tags['hostname']
ORDER BY hour
```

- QuestDB

```
SELECT
    timestamp,
    hostname,
    avg(usage_user) AS avg_usage_user,
    avg(usage_system) AS avg_usage_system,
     avg(usage_idle) AS avg_usage_idle,
     avg(usage_nice) AS avg_usage_nice,
     avg(usage_iowait) AS avg_usage_iowait
FROM
    cpu
WHERE
    timestamp >= '2016-02-18T01:34:28Z'
    AND timestamp < '2016-02-18T13:34:28Z'
SAMPLE BY 1h
GROUP BY timestamp, hostname
```

4) groupby-orderby-limit
Περιγραφή : Υπολογισμός των μέγιστων (μεταξύ όλων των hosts) cpu "usage_user" που παρατηρήθηκαν τα τελευταία 5 λεπτά πριν από κάποια καθορισμένη χρονική στιγμή
Υλοποίηση:

- CrateDB

```
SELECT
    date_trunc('minute', ts) as minute,
    max(usage_user)
FROM
    cpu
WHERE ts < 1453635272856
GROUP BY minute
ORDER BY minute DESC
LIMIT 5
```

- QuestDB

```
SELECT
    timestamp AS minute,
    max(usage_user)
FROM cpu
WHERE timestamp < '2016-01-24T11:34:32Z'
SAMPLE BY 1m
LIMIT 5
```

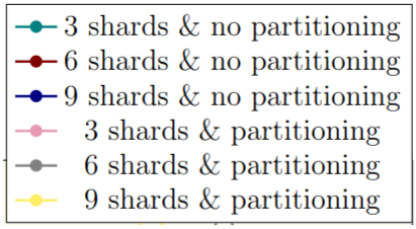
Τα παραπάνω queries επιλέχθηκαν καθώς περιέχουν aggregations , groupings, ordering και filtering για time intervals, στοιχεία πολύ χρήσιμα κατά την επεξεργασία δεδομένων πραγματικού χρόνου. Τα batches από queries που στέλνονται προς εκτέλεση περιλαμβάνουν 100 queries. Οι μετρήσεις

στην QuestDB γίνονται για 1, 2, 3, 4, 6 και 8 workers ενώ για την CrateDB μέχρι και για 18 workers και για διαφορετικά configurations της βάσης (3, 6, 9 shards με και χωρίς partitioning). Οι πολλαπλοί workers αναλαμβάνουν να στέλνουν ταυτόχρονα πολλαπλά requests προς το execution engine της εκάστοτε βάσης ώστε να την δοκιμάσουν στο πόσο καλά μπορεί να εκτελέσει παράλληλα διαφορετικά queries. Για κάθε είδος query, παρουσιάζονται 4 διαγράμματα : ένα για την QuestDB και 3 για την CrateDB, 1 εκ των οποίων είναι το συνολικό ενώ τα 2 τελευταία παρουσιάζουν μια decomposed μορφή του συνολικού διαγράμματος για λόγους ευκρίνειας. Ο legend των διαγραμμάτων της CrateDB είναι αυτός που φαίνεται στην εικόνα 20. Τα διαγράμματα των μετρήσεων φαίνονται στην εικόνα 22 της σελίδας 17 με βάση τα οποία κάνουμε κάποιες συγκρίσεις για κάθε query ξεχωριστά. Στην εικόνα 21 απεικονίζεται επίσης συγκεντρωτικά η επίδοση της QuestDB και η αντίστοιχη (κατά μέσο όρο) καλύτερη δυνατή επίδοση που μπορεί να επιτύχει η CrateDB (συγκρίνοντας τα διάφορα configurations της).

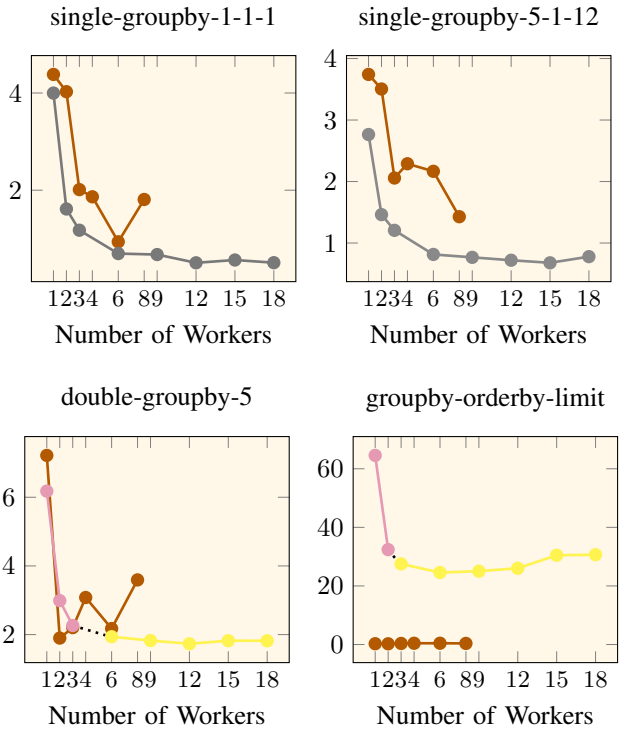
a) single-groupby-1-1-1: Παρατηρούμε ότι η CrateDB επιτυγχάνει την καλύτερη επίδοση συνολικά όταν χρησιμοποιεί 6 shards, είτε με είτε χωρίς partitioning. Με χρήση περισσότερων από 6 workers, η επίδοση σταθεροποιείται για κάθε configuration της βάσης. Η χρήση partitioning οδηγεί σε καλύτερες επιδόσεις (εξαίρεση η χρήση 9 shards ανά partition όπου τα shards γίνονται υπεράριθμα) καθώς τα queries περιορίζονται (μέσω του timestamp filtering) σε κάποια 1 ώρα ενός συγκεκριμένου μήνα, οδηγώντας το execution engine της Crate να διασχίσει μόνο το 1 partition (δηλαδή τελικά λιγότερα -και μικρότερα- shards). Τέλος, παρατηρούμε ότι η επίδοση της Quest είναι χειρότερη από την μέση επίδοση που μπορεί να επιτύχει η Crate εκτός από την περίπτωση του 1 worker, στην οποία στέλνεται κάθε φορά ένα request προς το εκάστοτε execution engine. Στην QuestDB το κάθε query όταν εκτελείται ατομικά, ολοκληρώνεται ταχύτερα καθώς στην CrateDB απαιτεί δικτυακή επικοινωνία και συνεργασία των διαφορετικών nodes. Ωστόσο, όταν στις δύο βάσεις στέλνονται ταυτόχρονα πολλαπλά requests για εκτέλεση queries, η CrateDB επιτυγχάνει καλύτερες επιδόσεις λόγω της κατανεμημένης φύσης τη, με φράγμα τη χρήση 6+ workers, σημείο πάνω από το οποίο η Crate φαίνεται να ξεπερνά το όρια της στην παράλληλη ικανοποίηση requests για αυτό το είδος query ανεξαρτήτως configuration .Η shared nothing αρχιτεκτονική της, η οποία εξασφαλίζει την ισοτιμία των κόμβων ως προς την ανάληψη tasks και ικανοποίηση αιτημάτων, ενισχύει τον παραλληλισμό οδηγώντας στα παραπάνω αποτελέσματα. Σε κάθε περίπτωση, συγκρίνοντας την επίδοση της Quest με την καλύτερη επίδοση που μπορεί να επιτύχει η Crate (6 shards με partitioning), παρατηρούμε ότι η Crate υπερισχύει.

b) single-groupby-5-1-12: Η CrateDB σε αυτή την περίπτωση, επιτυγχάνει καλύτερη επίδοση συνολικά όταν χρησιμοποιεί 6 shards με partitioning, ενώ γενικά η χρήση partitioning οδηγεί σε καλύτερες επιδόσεις από την μη χρήση, για τον ίδιο λόγο με πριν. Η καλύτερη επίδοση της CrateDB (grey line) είναι καλύτερη από την επίδοση της QuestDB, ως προς την μέση περίπτωση η χρήση ενός worker ευνοεί την Quest αντί της Crate για τον λόγο που εξηγήθηκε επίσης προηγουμένως, ενώ με χρήση 6+ workers η CrateDB φαίνεται ότι δεν μπορεί να ανταποκριθεί σε αυτό το μέγεθος παραλληλισμού στην ικανοποίηση requests για αυτό το είδος query.

c) double-groupby-5: Σε ό,τι αφορά την επίδοση της Crate σε αυτήν την περίπτωση, παρατηρούμε ότι μέχρι 3 concurrent TSBS workers, υπερισχύουν τα configurations της Crate που περιλαμβάνουν πολλά shards (σε αντίθεση με ό,τι παρατηρήθηκε στα προηγούμενα queries), ενώ αντιθέτως με πολλά ταυτόχρονα requests (6+) -σε αντίθεση με πριν οπότε και η επίδοση σταθεροποιούταν- υπάρχουν διαφοροποιήσεις στην επίδοση και συγκεκριμένα ευνοούνται τα configurations της Crate που περιλαμβάνουν λιγότερα shards. Όταν έχουμε

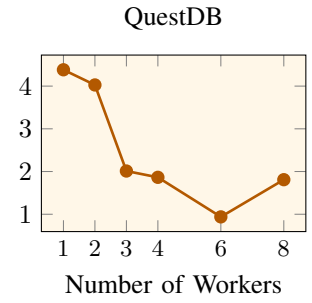


Εικόνα 20: CrateDB Legend

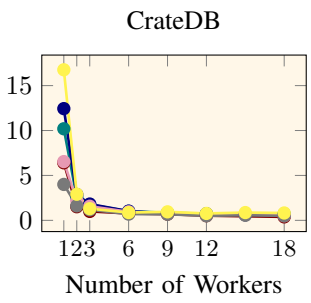


Εικόνα 21: Συγκριτική παρουσίαση της επίδοσης της QuestDB και της εκάστοτε καλύτερης επίδοσης της CrateDB ανά τύπο TSBS query batch

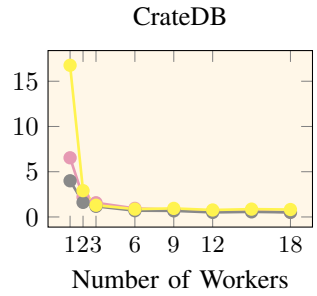
single-groupby-1-1-1



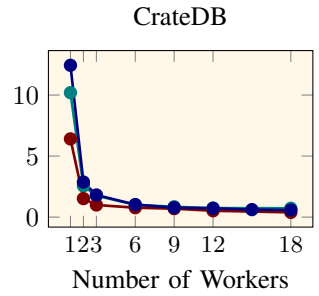
(a) Time per Num of Workers QuestDB-1st query



(b) Time per Num of Workers CrateDB-1st query

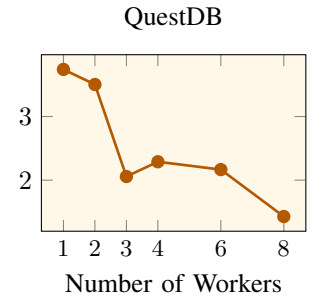


(c) CrateDB Decomposed 1st Query-With Partitioning

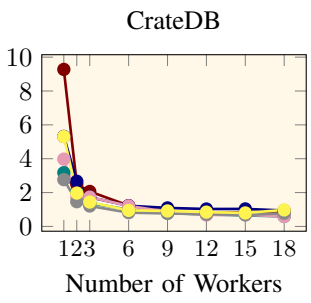


(d) CrateDB Decomposed 1st Query-Without Partitioning

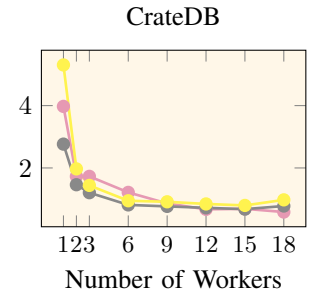
single-groupby-5-1-12



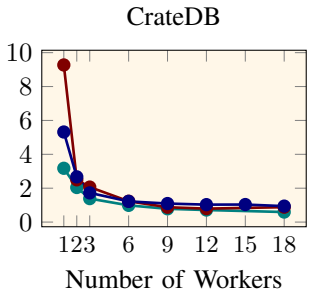
(e) Time per Num of Workers QuestDB-2nd query



(f) Time per Num of Workers CrateDB-2nd query

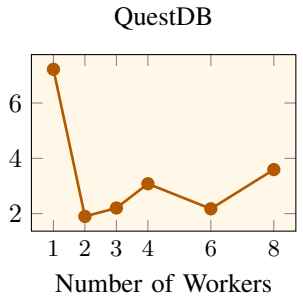


(g) CrateDB Decomposed 2nd Query-With Partitioning

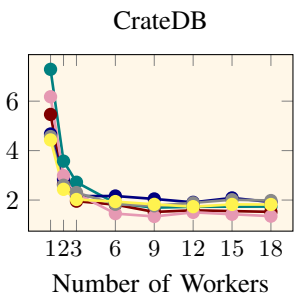


(h) CrateDB Decomposed 2nd Query-Without Partitioning

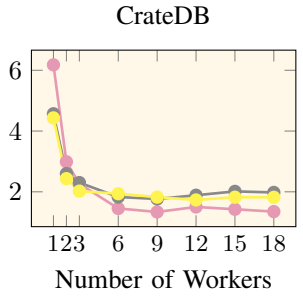
double-groupby-5



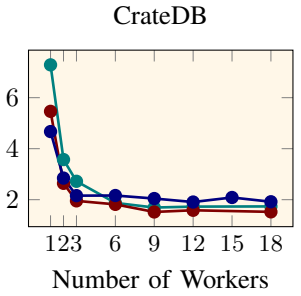
(i) Time per Num of Workers QuestDB-3rd query



(j) Time per Num of Workers CrateDB-3rd query

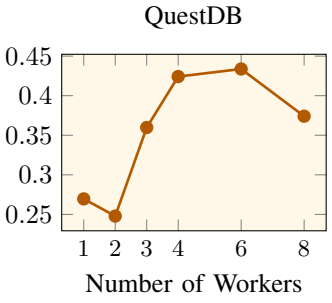


(k) CrateDB Decomposed 3rd Query-With Partitioning

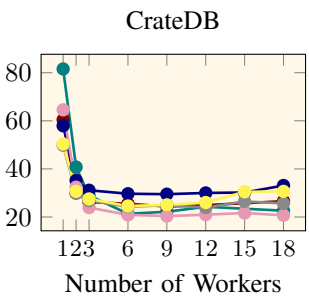


(l) CrateDB Decomposed 3rd Query-Without Partitioning

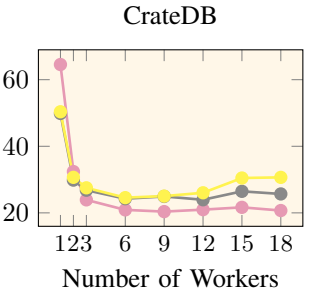
groupby-orderby-limit



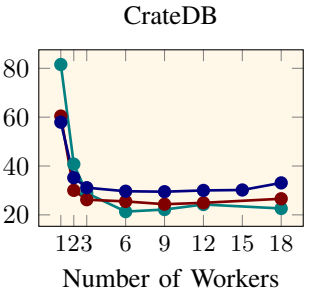
(m) Time per Num of Workers QuestDB-4th query



(n) Time per Num of Workers CrateDB-4th query



(o) CrateDB Decomposed 4th Query-With Partitioning



(p) CrateDB Decomposed 4th Query-Without Partitioning

Εικόνα 22: Χρόνοι Εκτέλεσης batches από TSBS Queries με διάφορους TSBS workers και CrateDB configurations

λίγα παράλληλα requests (workers), το γεγονός ότι το παρόν query είναι πιο απαιτητικό (διπλό group by), τα πολλά μικρότερα shards οδηγούν την εκτέλεση των δύσκολων σταδίων group by σε ένα μικρότερο shard (μετά το filtering τελικά η χρήσιμη πληροφορία 12 ωρών του κάθε query βρίσκεται σε ένα μόνο shard) συγκριτικά με την περίπτωση στην οποία τα shards θα ήταν λιγότερα αλλά μεγαλύτερα. Ενώ αυτό το χαρακτηριστικό θα μπορούσε να είχε ευνοήσει και το single-groupby-1-1-1, αυτό ειναι αρκετά πιο απλό και η ύπαρξη πολλών shards τελικά οδηγεί στην επικράτηση του διαχειριστικού κόστους τους παρά σε ό,τι αναφέρθηκε προηγουμένως. Όταν έχουμε πολλά παράλληλα requests (workers), αφενός η επίδοση δεν σταθεροποιείται καθώς το πιο πολύπλοκο query ευνοείται από την ύπαρξη παραλληλισμού αφετέρου η επίδοση βελτιώνεται με χρήση λιγότερων (και άρα μεγαλύτερων) shards καθώς είναι προτιμότερο να συμβαίνει contention μεταξύ των queries σε μεγαλύτερα παρά σε μικρότερα shards. Ομοίως, η καλύτερη επίδοση της CrateDB (τόσο με πολλούς όσο και με λίγους TSBS workers) είναι καλύτερη από την επίδοση της Quest. Εδώ, σε αντίθεση με τις προηγούμενες περιπτώσεις, η χρήση ενός TSBS worker οδηγεί σε παρόμοιες μέσες επιδόσεις στις δύο βάσεις. Τώρα, το γεγονός ότι το query είναι πιο περίπλοκο, το κατανεμημένο περιβάλλον της Crate συνεισφέρει θετικά καθώς δεν κυριαρχούν τα overheads δικτυακής επικοινωνίας των nodes, όπως στα προηγούμενα queries.

d) groupby-orderby-limit: Καταρχάς η πιο σημαντική παρατήρηση εδώ είναι ότι η QuestDB έχει εντυπωσιακά καλύτερη επίδοση από αυτήν της CrateDB σε κάθε περίπτωση. Όπως παρατηρούμε στην υλοποίηση των δύο queries, η Quest αποφεύγει τη χρήση των GROUP BY, ORDER BY και τα αντικαθιστά με την ειδικά υλοποιημένη εντολή SAMPLE BY, η οποία φαίνεται ότι επιτυγχάνει πολύ καλύτερες επιδόσεις. Ωστόσο δεν αρκεί αυτό διότι παρόμοιες υλοποιήσεις (με τη βοήθεια της SAMPLE BY) είχαν και τα προηγούμενα queries. Ένα βασικό όμως χαρακτηριστικό της Quest είναι ότι διατηρεί στην cache αποτελέσματα προηγούμενων queries (κάτι που παρατηρήθηκε πολύ έντονα κατά τους πειραματισμούς μας στην παρούσα εργασία). Ως εκ τούτου, επειδή το συγκεκριμένο είδος query έχει ένα πολύ εκτενές time interval στο where condition, επόμενα queries του batch τυγχάνει να χρησιμοποιούν time intervals επικαλυπτόμενα με αυτά των προηγούμενων queries επωφελούμενα από τη χρήση της cache. Σε ό,τι αφορά στην συμπεριφορά της CrateDB για τα διαφορετικά configurations και TSBS workers, είναι παρόμοια με αυτήν του προηγούμενου query καθώς και το παρόν query είναι πολύ απαιτητικό. Τέλος, κατά απόλυτη τιμή χρόνων η Crate φαίνεται ότι λόγω των μεγάλων intervals που διασχίζουν αυτά τα queries, απαιτεί σημαντικό χρόνο για την εκτέλεση.

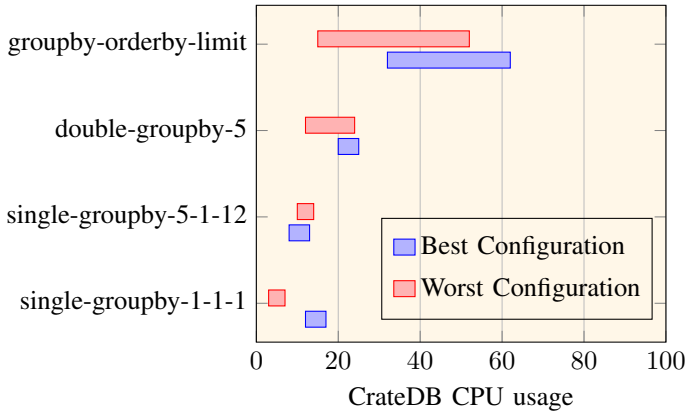
Τέλος, σημειώνουμε για την συμπεριφορά της QuestDB ότι η επίδοσή της βελτιώνεται με αύξηση των TSBS workers μέχρι ένα σημείο μετά το οποίο εν γένει η επίδοση επιδεινώνεται, καθώς η Quest αδυνατεί να εξυπηρετήσει περισσότερα παράλληλα requests. Ακόμη, η επίδοση της Quest ευνοείται από την χρήση κατάλληλων SQL extensions, όπως το SAMPLE BY (που κάνει grouping των δεδομένων ανά σταθερό χρονικό βήμα π.χ. ανά λεπτό) αλλά και από την χρήση indexes σε περιπτώσεις attributes τύπου SYMBOL, όπως το "hostname" (υπενθυμίζεται ότι η QuestDB χρησιμοποιεί indexes μόνο για τα columns τύπου SYMBOL). Σε κάθε περίπτωση, η CrateDB φαίνεται να διαχειρίζεται καλύτερα batches από queries (ειδικά όταν αυτά στέλνονται παράλληλα), με εξαίρεση την περίπτωση στην οποία αυτά τα queries εμφανίζουν ομοιότητες που ευνοούνται από το caching της QuestDB.

B. Μετρικές επίδοσης κατά την εκτέλεση *batches* από *TSBS queries*

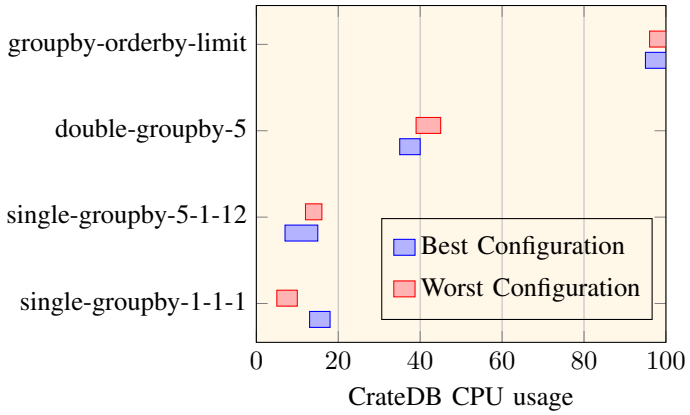
Στη συνέχεια, παρουσιάζονται κάποιες μετρικές επίδοσης (cpu usage, average query latency, throughput) που παρατηρήθηκαν κατά την εκτέλεση των παραπάνω batches από queries. Αρχικά, παρουσιάζεται για την CrateDB (για την QuestDB δεν παρέχονται τέτοιες πληροφορίες από το UI της) το usage της cpu όπως αυτό μετράται κατά την εκτέλεση των τεσσάρων διαφορετικών query batches. Στις εικόνες 24 και 25 παρουσιάζονται αυτές οι μετρήσεις. Ο Master node και ο worker 1 είχαν παρόμοια συμπεριφορά σε αντίθεση με τον worker 2 που παρουσιάζε σε κάποιες περιπτώσεις αρκετά διαφορετική συμπεριφορά. Για αυτό το λόγο παρουσιάζονται δύο διαγράμματα, το πρώτο για τον

```
single-groupby-1-1-1:
    Best: 6 shards with partitioning
    Worst: 9 shards without partitioning

single-groupby-5-1-12:
    Best: 6 shards with partitioning
    Worst: 6 shards without partitioning

Double-groupby-5:
    Best:
        up to 3 workers:
            9 shards with partitioning
        > 3 workers :
            3 shards with partitioning
    Worst:
        up to 3 workers:
            3 shards without partitioning
        > 3 workers:
            9 shards without partitioning

Groupby-orderby-limit:
    Best:
        up to 2 workers:
            9 shards with partitioning
        > 2 workers:
            3 shards with partitioning
    Worst:
        up to 2 workers:
            3 shards without partitioning
        > 2 workers:
            9 shards with partitioning
```

Εικόνα 23: Καλύτερες και χειρότερες (ως προς τον απαιτούμενο χρόνο εκτέλεσης) περιπτώσεις configuration της CrateDB ανά τύπο query



Εικόνα 24: CrateDB CPU usage intervals (%) for Master and Worker1 nodes and for the best and worst configuartion scenarios (as far as the execution time concerned)



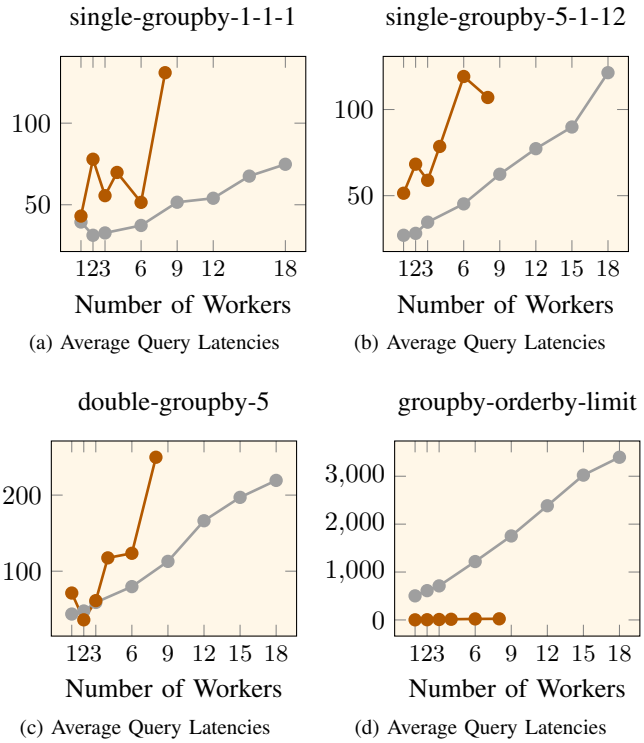
Εικόνα 25: CrateDB CPU usage intervals (%) for Worker2 node and for the best and worst configuartion scenarios (as far as the execution time concerned)

Master και worker 1 και το δεύτερο για τον worker 2. Σε κάθε ένα από αυτά τα διαγράμματα, δίνονται μετρήσεις για δύο configurations (δηλαδή πλήθος από shards και ύπαρξη ή μη partitioning), αυτό με το οποίο επετεύχθει ο μικρότερος κατά μέσο όρο χρόνος (best configuration) κατά την εκτέλεση του αντίστοιχου είδους query και αυτό με το οποίο παρατηρήθηκε ο μεγαλύτερος χρόνος (worst configuration). Οι πληροφορίες για το ποιο configuration είναι καλύτερο ή χειρότερο ανά περίπτωση query φαίνεται στην εικόνα 23. Για κάθε τύπο query, λοιπόν παρουσιάζεται για αυτές τις δύο ακραίες περιπτώσεις (best, worst) διαστήματα τιμών που έλαβε το cpu usage κατά τη διάρκεια εκτέλεσης batches από 100 τέτοια queries.

Παρατηρούμε από τις εικόνες 24 και 25 τα παρακάτω :

- Στην πλειοψηφία των περιπτώσεων, το cpu usage στο best configuration είναι κατά μέσο όρο μεγαλύτερο από αυτό στο worst configuration. Αυτό θα μπορούσε να σημαίνει ότι το γεγονός πως στο worst configuration επιτυγχάνονται χειρότερες επιδόσεις οφείλεται σε underutilization της cpu ενώ ο χρόνος ενδέχεται να δαπανάται -σε σημαντικό βαθμό- σε μη ωφέλιμες πράξεις (π.χ. επικοινωνία μεταξύ των κόμβων). Αντιθέτως στο best configuration, το σύστημα αξιοποιεί περισσότερο την cpu, οδηγώντας σε μικρότερους χρόνους εκτέλεσης.
- Αν εξαιρέσουμε το query groupby-orderby-limit, οι διακυμάνσεις των τιμών του cpu usage είναι μικρές και μάλιστα κυμαίνονται σε σχετικά αποδεκτά επίπεδα (κάτω του 50%). Αντιθέτως, στο query batch groupby-orderby-limit εκτός του γεγονότος ότι στον Master και worker 1 το cpu usage παρουσιάζει σημαντικές διακυμάνσεις (μεταξύ του 20% και 60%), στον worker 2 φθάνει να χρησιμοποιεί πλήρως την cpu (usage 95% με 100%), δημιουργώντας ένα πιθανό bottleneck σε αυτό το σημείο. Εξάλλου το γεγονός ότι η Crate δυσκολεύεται στην εκτέλεση αυτού του query φάνηκε και από τον μεγάλο χρόνο εκτέλεσης που απαιτήθηκε (εικόνα 22). Πιθανή βελτίωση των χαρακτηριστικών των μηχανημάτων μας, θα μπορούσε συνεπώς θα επιλύσει τέτοια bottlenecks (π.χ. μηχανήματα με 8 και όχι 4 CPU cores).

Στη συνέχεια, παρουσιάζεται στην εικόνα 26 το μέσο latency για την εκτέλεση ενός query όταν στέλνονται batches από 100 τέτοια queries με χρήση πολλαπλών TSBS workers. Αυτή η μετρική παρουσιάζεται για την QuestDB και για

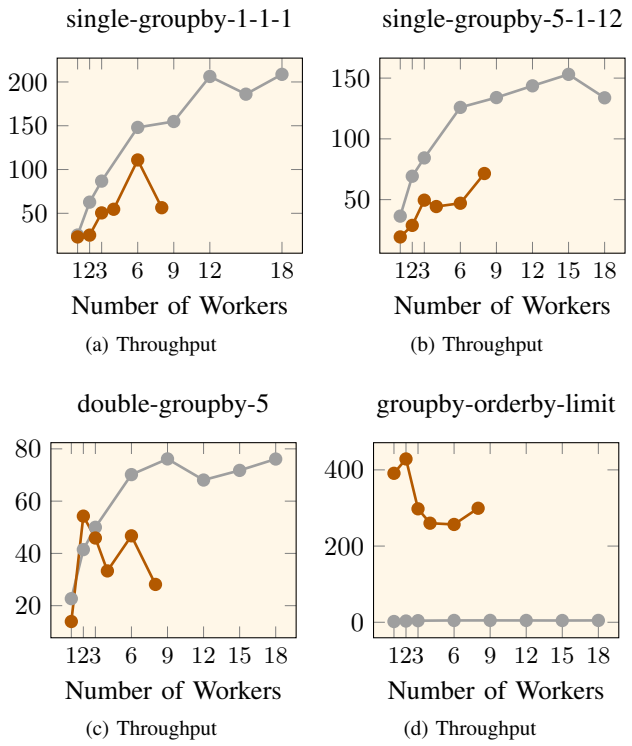


(a) Average Query Latencies (b) Average Query Latencies

(c) Average Query Latencies (d) Average Query Latencies

Εικόνα 26: Μέσα query latencies για κάθε τύπο query μεταξύ της QuestDB και της καλύτερης περίπτωσης της CrateDB (με γκρίζο χρώμα η CrateDB).

την καλύτερη περίπτωση της CrateDB και υπολογίζεται αθροίζοντας τους χρόνους εκτέλεσης καθενός από τα 100 queries και διαιρώντας με 100 (Προσοχή: δεν ισούται με τον συνολικό χρόνο εκτέλεσης δια 100 καθώς αυτός δεν είναι το άθροισμα των επιμέρους χρόνων αφού αυτοί επικαλύπτονται όταν χρησιμοποιούνται πολλαπλοί TSBS workers για αποστολή αιτημάτων). Αυτή η μετρική ουσιαστικά αναδεικνύει τον μέσο χρόνο κατά τον οποίο θα χρειαστεί να περιμένει ένας χρήστης όταν στέλνει ένα αίτημα (query) προς το σύστημα. Η συμπεριφορά είναι παρόμοια με αυτήν που παρατηρήθηκε στην εικόνα 21 για τους χρόνους εκτέλεσης. Η CrateDB επικρατεί της QuestDB με εξαίρεση το query groupby-orderby-limit όπου η QuestDB επιτυγχάνει εντυπωσιακές επιδόσεις όπως αναφέρθηκε παραπάνω.
Τέλος, παρουσιάζεται και ένα διάγραμμα με το throughput (πλήθος από queries που ικανοποιούνται ανά μονάδα χρόνου) που επιτυγχάνεται ανά περίπτωση [εικόνα 27]. Τα συμπεράσματα είναι εν γένει ίδια με αυτά του query latency.



(a) Throughput (b) Throughput (c) Throughput (d) Throughput

Εικόνα 27: Throughput για την QuestDB και για το εκάστοτε καλύτερο configuration της CrateDB για κάθε τύπο query. Με γκρίζο χρώμα, η CrateDB.

C. Χρήση *custom queries*

Στη συνέχεια, εξετάζουμε την επίδοση (ως προς τον χρόνο εκτέλεσης) που επιτυγχάνεται από την QuestDB και από την CrateDB (με διάφορα configurations) σε ενδεικτικά queries που διαμορφώθηκαν χωρίς χρήση του TSBS, καθώς θεωρήθηκαν ενδιαφέροντα για την ανάδειξη διάφορων χαρακτηριστικών των δύο βάσεων. Τα queries που θα χρησιμοποιηθούν όπως και η υλοποίησή τους στις δύο βάσεις φαίνονται παρακάτω. Η διαφορά στις υλοποιήσεις μεταξύ των δύο βάσεων είναι συνήθως μικρές (εκτός από την περίπτωση του query 2 όπως θα εξηγηθεί). Σημειώνουμε ότι σε αυτό το μέρος, δεν κάνουμε μετρήσεις για batches από τέτοια queries αλλά για την ατομική και ξεχωριστή εκτέλεση καθενός από αυτά.

1) Παρουσίαση και περιγραφή των χρησιμοποιούμενων *queries:*

a) Query 1: Κώδικες σε SQL

```
//QuestDB:
SELECT * FROM nginx LIMIT 100000

//CrateDB:
SELECT * FROM benchmark.nginx
LIMIT 100000
```

Περιγραφή : Εμφάνιση των πρώτων 100000 στοιχείων ενός εκ των πινάκων (απλό SELECT).

b) Query 2: Κώδικες σε SQL

```
//QuestDB:
SELECT
    c.usage_user usage_user,
    d.used_percent
FROM
    cpu as c
ASOF JOIN disk as d;

//CrateDB:
SELECT
    c.usage_user usage_user,
    a.used_percent
FROM
    (SELECT * FROM benchmark.cpu LIMIT 10000)
        as c
JOIN (
    SELECT
        ts,
        tags['hostname'],
        used_percent
    FROM
    (SELECT * FROM benchmark.disk LIMIT 10000)
        as d
) a
ON
    c.ts >= a.ts
    and
    c.tags['hostname']=a.tags['hostname']
ORDER BY
    c.ts, a.ts DESC;
```

Περιγραφή : Χρήση του ASOF JOIN (που υλοποιείται στην QuestDB) και της αντίστοιχης πιο αναλυτικής υλοποίησης που αναγκαστικά χρησιμοποιείται στην CrateDB (που δεν υλοποιεί το ASOF JOIN). Η συγκεκριμένη πράξη χρησιμοποιείται όταν θέλουμε να κάνουμε join δύο πινάκων με βάση το timestamp. Μιας και τα timestamps των δύο πινάκων είναι απίθανο να είναι ακριβώς ίδια σε πραγματικές συνθήκες, το συγκεκριμένο SQL extension επιτρέπει δύο timestamps να θεωρηθούν ίσα αν είναι επαρκώς κοντά το ένα με το άλλο χρονικά (συγκεκριμένα το timestamp του ενός table αντιστοιχίζεται στο ακριβώς μικρότερο και πλησιέστερο χρονικά timestamp του δεύτερου table).

c) Query 3: Κώδικες σε SQL

```
//QuestDB:
SELECT
```

```
    hostname,
    timestamp,
    usage_user
FROM
    cpu
WHERE
    timestamp in '2016-02'

//CrateDB:
SELECT
    tags['hostname'],
    ts,
    usage_user
FROM
    benchmark.cpu
WHERE
    ts<'2016-03-01T00:00:00.000000Z'
    and
    ts>='2016-02-01T00:00:00.000000Z'
limit 100000;
```

Περιγραφή: Πρόκειται για ένα range query που λαμβάνει δεδομένα από έναν ολόκληρο μήνα (Φεβρουάριο). Δεδομένου του partitioning by month που μεταξύ άλλων εφαρμόζεται παρακάτω στις δύο βάσεις, αυτό θα σημαίνει ότι αυτό το query αποκτά πρόσβαση σε ένα συγκεκριμένο partition, με ό,τι αυτό μπορεί να συνεπάγεται στην επίδοση (όπως αυτό θα εξηγηθεί αργότερα).

d) Query 4: Κώδικες σε SQL

```
//QuestDB:
SELECT
    hostname,
    timestamp,
    usage_user
FROM
    cpu
WHERE
    timestamp<'2016-02-15T00:00:00.000000Z'
    and
    timestamp>='2016-01-15T00:00:00.000000Z'

//CrateDB:
SELECT
    tags['hostname'],
    ts,
    usage_user
FROM
    benchmark.cpu
WHERE
    ts<'2016-02-15T00:00:00.000000Z'
    and
    ts>='2016-01-15T00:00:00.000000Z'
limit 100000;
```

Περιγραφή: Πρόκειται για ένα range query που λαμβάνει δεδομένα ενός μήνα, από τα μέσα Ιανουαρίου έως τα μέσα Φεβρουαρίου. Αυτό (για σύγκριση με το προηγούμενο query) θα σημαίνει ότι αυτό το query αποκτά πρόσβαση σε δύο διαφορετικά partitions, οδηγώντας σε διαφοροποίηση ως προς την επίδοση, όπως θα παρουσιαστεί παρακάτω.

e) Query 5: Κώδικες σε SQL

```
//QuestDB:
SELECT
    first.hostname,
    total_packets_recv
FROM
(
    SELECT hostname, maximum,minimum
    FROM
    (
        select
            hostname,
            max(used_percent)
                as maximum,
            min(used_percent)
                as minimum,
            max(os)
                as OS
        from mem
        WHERE
            timestamp>='2016-01-01T00:00:00.000000Z'
            AND
            timestamp<='2016-02-27T00:00:00.000000Z'
        GROUP BY hostname
    )
WHERE maximum<=1.1*minimum
) as first
JOIN
(
SELECT
    hostname,
    sum(packets_recv)
        as total_packets_recv
    FROM
        net
WHERE
    timestamp>='2016-01-01T00:00:00.000000Z'
    AND
    timestamp<='2016-02-27T00:00:00.000000Z'
GROUP BY hostname
) as second
ON
    first.hostname=second.hostname
ORDER BY
    first.maximum DESC

//CrateDB:
SELECT
    one.tags['hostname'],
    total_packets_recv
FROM
(
    SELECT tags['hostname'], maximum,minimum
    FROM
    (
        select
            tags['hostname'],
            max(used_percent)
                as maximum,
            min(used_percent)
                as minimum,
            max(tags['os'])
                as OS
        from benchmark.mem
        WHERE
            ts>='2016-01-01T00:00:00.000000Z'
            AND
        ts<='2016-02-27T00:00:00.000000Z'
        GROUP BY tags['hostname']
    ) as nam
    WHERE
        maximum<=1.1*minimum
) as one
JOIN
(
    SELECT
        tags['hostname'],
        sum(packets_recv)
            as total_packets_recv
        FROM
            benchmark.net
        WHERE
            ts>='2016-01-01T00:00:00.000000Z'
```

```
            AND
            ts<='2016-02-27T00:00:00.000000Z'
        GROUP BY tags['hostname']
) as two
ON
    one.tags['hostname']=two.tags['hostname']
ORDER BY
    one.maximum DESC
LIMIT 100000;
```

Περιγραφή: Πρόκειται για ένα query που υλοποιεί ένα πιθανό σενάριο χρήσης των βάσεων. Επιλέγει τους hosts που σε ένα συγκεκριμένο time interval είχαν σχεδόν σταθερό memory usage (η μέγιστη τιμή ήταν το πολύ 10% πάνω από την ελάχιστη). Για αυτούς, βρίσκει για το ίδιο time interval το πλήθος των συνολικών πακέτων που παρέλαβαν και τους κατατάσσει κατά φθίνουσα σειρά μέγιστης χρήσης της memory. Περιλαμβάνει JOIN μεταξύ δύο tables, ORDER BY, GROUP BY και time interval στο WHERE clause.

f) Query 6: Κώδικες σε SQL

```
//QuestDB:
SELECT
    first.hostname,
    first.timestamp,
    first.usage_user,
    second.used_percent ,
    k.context_switches
FROM
(
    SELECT
        hostname,
        usage_user,
        timestamp
    FROM
        cpu
    WHERE
        usage_user>0
    LIMIT 10000
) as first
JOIN
(
    SELECT
        hostname,
        used_percent,
        timestamp
    FROM
        mem
) as second
ON
    first.hostname=second.hostname
    AND
    first.timestamp=second.timestamp
    AND
    second.used_percent>=10*first.usage_user
INNER JOIN
    kernel as k
ON
    second.hostname=k.hostname
    AND
    k.timestamp=second.timestamp


//CrateDB:
SELECT
    one.tags['hostname'],
    one.ts,
    one.usage_user,
    two.used_percent ,
    k.context_switches
FROM
(
    SELECT
        tags['hostname'],
        usage_user,
        ts
    FROM
        benchmark.cpu
    WHERE
        usage_user>0
    LIMIT 10000
) as one
JOIN
(
    SELECT
        tags['hostname'],
        used_percent,
        ts
    FROM
        benchmark.mem
    LIMIT 10000
) as two
ON
    one.tags['hostname']=two.tags['hostname']
    AND
    one.ts=two.ts
    AND two.used_percent>=10*one.usage_user
INNER JOIN
    benchmark.kernel as k
ON
    two.tags['hostname']=k.tags['hostname']
AND
    k.ts=two.ts
```

Περιγραφή: Πρόκειται για ένα δεύτερο σενάριο χρήσης των βάσεων. Βρίσκει τους hosts που κάποια (ή κάποιες) χρονική στιγμή πέτυχαν memory used_percent (ποσοστό χρήσης της μνήμης) τουλάχιστον δεκαπλάσιο του (μη μηδενικού) cpu usage_user (χρησιμοποίηση της cpu) και για την ίδια χρονική στιγμή υπολογίζει πόσα context switches είχαν κάνει οι συγκεκριμένοι hosts. Αυτό το query περιλαμβάνει JOIN μεταξύ τριών tables.

g) Query 7: Κώδικες σε SQL

```
//QuestDB:
SELECT hostname, requests
FROM nginx
WHERE timestamp='2016-01-18T17:42:00.000000Z';

//CrateDB:
SELECT tags['hostname'], requests
FROM benchmark.nginx
WHERE ts='2016-01-18T17:42:00.000000Z'
```

Περιγραφή: Πρόκειται για ένα point query που κάνει retrieve πληροφορίες σχετικές με μια συγκεκριμένη χρονική στιγμή.

h) Query 8: Κώδικες σε SQL

```
//QuestDB:
SELECT hostname, max(used_percent) as maxUsedPer
FROM disk
GROUP BY hostname;

//CrateDB:
SELECT tags['hostname'], max(used_percent) as
    maxUsedPer
FROM benchmark.disk
GROUP BY tags['hostname'];
```

Περιγραφή: Πρόκειται για query που χρησιμοποιεί μια απλή πράξη GROUP BY και κάποιο aggregation.

i) Query 9: Κώδικες σε SQL

```
//QuestDB:
SELECT a.hostname, a.used_percent,timestamp,a.
    ranking FROM (SELECT hostname as hostname,
    used_percent,timestamp,ROW_NUMBER() OVER (
    PARTITION BY hostname ORDER BY used_percent DESC
    ) AS ranking FROM mem)AS a WHERE a.ranking<4
    ORDER BY a.hostname,ranking ASC

//CrateDB:
SELECT a.hostname, a.used_percent,ts,a.ranking FROM
    (SELECT tags['hostname'] as hostname,
    used_percent,ts,ROW_NUMBER() OVER (PARTITION BY
    tags['hostname'] ORDER BY used_percent DESC) AS
    ranking FROM benchmark.mem)As a WHERE a.ranking
    <4 ORDER BY a.hostname,ranking ASC
```

Περιγραφή: Σενάριο που περιλαμβάνει window function (row_number()). Βρίσκει για κάθε host τα 3 μεγαλύτερα memory usages που πέτυχαν (δίνοντας σε αυτά ένα ranking) και τις χρονικές στιγμές στις οποίες τα πέτυχαν. Οι πληροφορίες εμφανίζονται ανά host σε αύξουσα σειρά ranking (απλώς για λόγους ευκρίνειας των αποτελεσμάτων και όχι για εξέταση της επίδοσης της πράξης ORDER BY, καθώς αυτή τελικά εκτελείται πολύ γρήγορα σε ένα σύνολο από 20 μόνο rows)

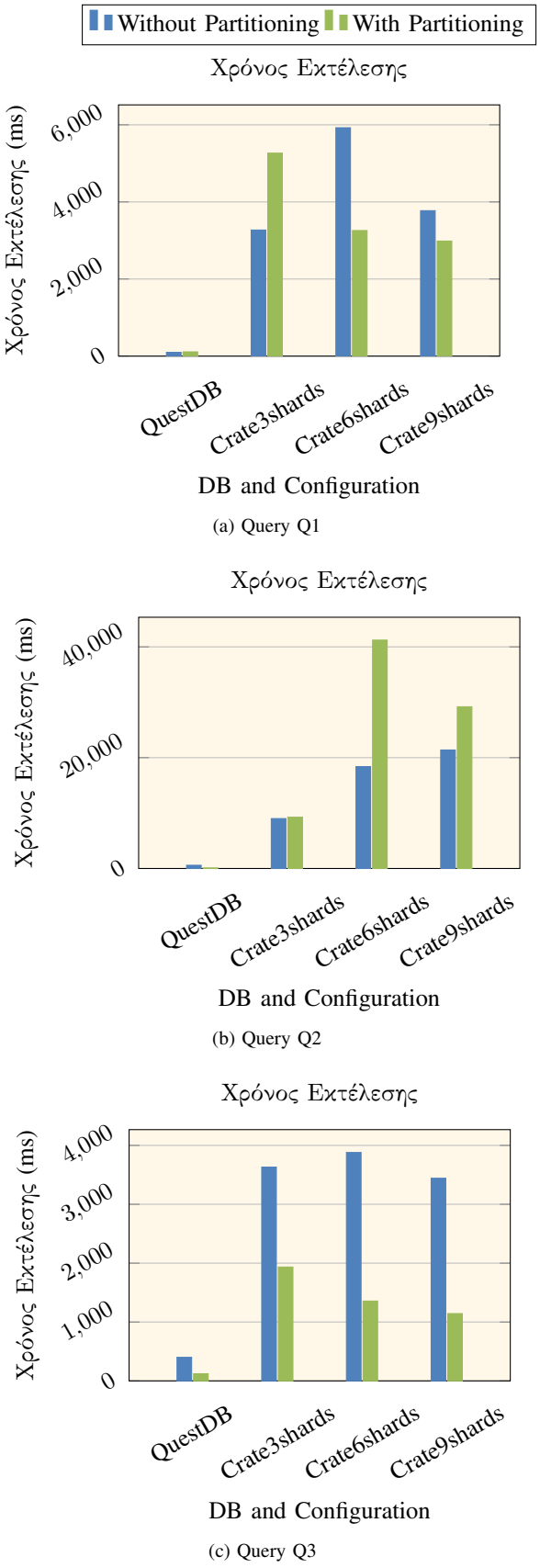
2) Ανάλυση επιδόσεων: Στη συνέχεια, εκτελούμε τα παραπάνω queries στην QuestDB (με και χωρίς partition) και στην CrateDB για όλα τα configurations (3, 6 ,9 shards με και χωρίς partition). Για την προσθήκη μηνιαίου partitioning στην QuestDB, τροποποιούμε το configuration file της, server.conf προσθέτοντας το εξής option:

```
line.default.partition.by=MONTH
```

διότι διαφορετικά η QuestDB δουλεύει by default με partitioning by day (και αυτό στην περίπτωσή μας οδηγεί σε πολύ μεγάλο αριθμό από partitions). Τα αποτελέσματα των μετρήσεων φαίνονται στα διαγράμματα της εικόνας 28, στη συνέχεια. Παρουσιάζονται ακολούθως οι παρατηρήσεις μας ανά query.

a) Query 1: Η QuestDB επιτυγχάνει πολύ καλύτερες επιδόσεις (χωρίς η επίδοσή της να βελτιώνεται με χρήση partition), ενώ στη γενική περίπτωση η CrateDB βελτιώνει την επίδοσή της στο SELECT όταν υπάρχουν συνολικά περισσότερα shards (που ισούνται με τον αριθμό shards ανά partition -3,6 ή 9- επί το πλήθος των partitions, εδώ 2). Έτσι, για παράδειγμα η χρήση 3 shards με partition (άρα 6 shards) έχει παρόμοιες επιδόσεις με τη χρήση 6 shards χωρίς partition και οι δύο είναι χειρότερες από τη χρήση 9 shards χωρίς partition που είναι χειρότερη από τη χρήση 6 shards με partition (12 shards) κ.ο.κ. Εξαίρεση αποτελεί η μετάβαση από τα 3 shards χωρίς partition στα 3 με partition όπου σε αντίθεση με την γενική συμπεριφορά, οδηγούμαστε σε χειρότερες επιδόσεις.

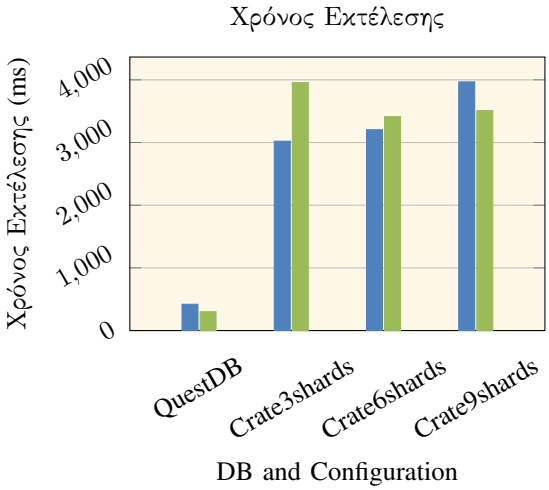
b) Query 2: Η χρήση του ειδικά υλοποιημένου ASOF JOIN από την QuestDB την οδηγεί σε εντυπωσιακά καλύτερες επιδόσεις συγκριτικά με την CrateDB, η οποία υλοποιεί την ίδια λειτουργία αλλά με μια περιγραφική και όχι τόσο αποδοτική υλοποίηση. Ενώ η χρήση partitioning στην QuestDB βελτιώνει ακόμη περισσότερο την επίδοση



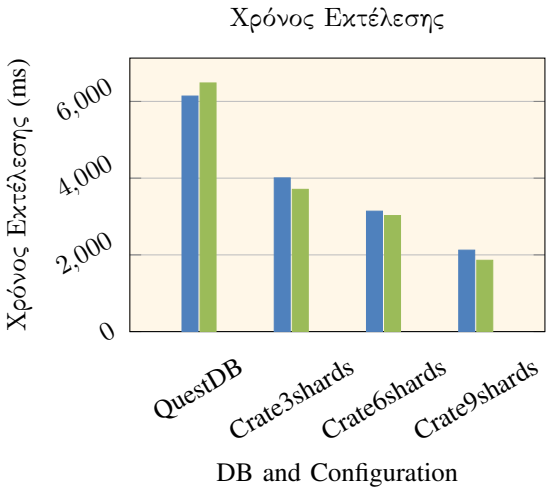
(a) Query Q1

(b) Query Q2

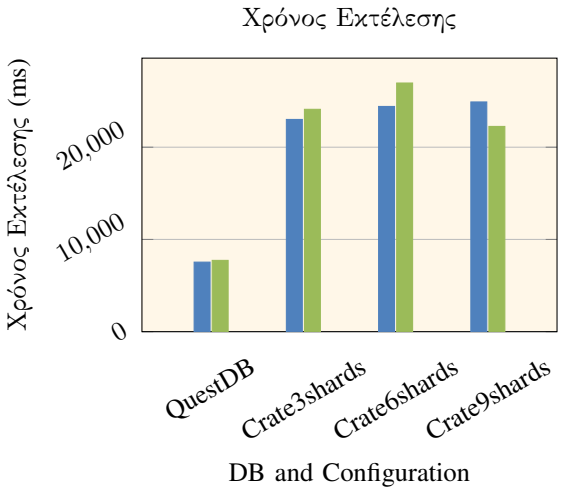
(c) Query Q3



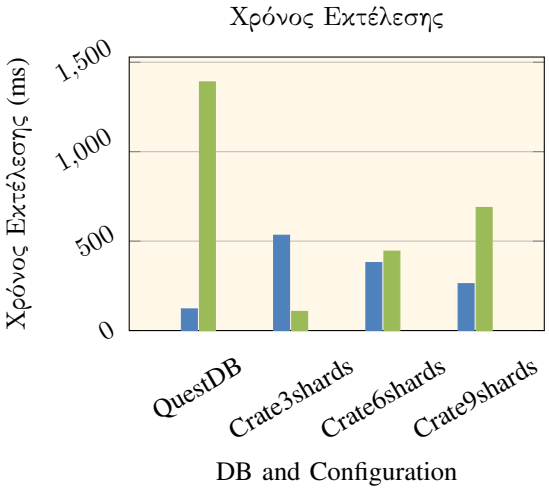
(d) Query Q4



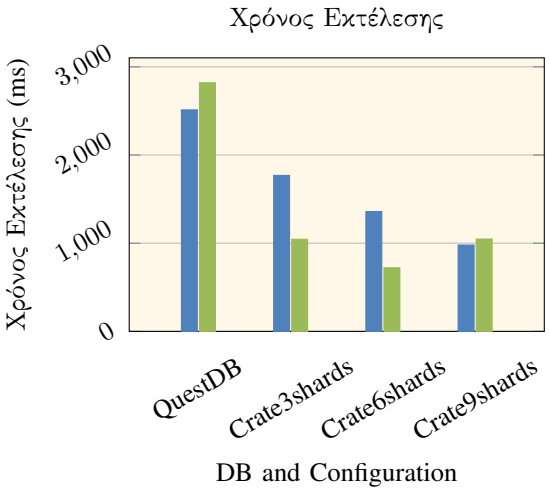
(e) Query Q5



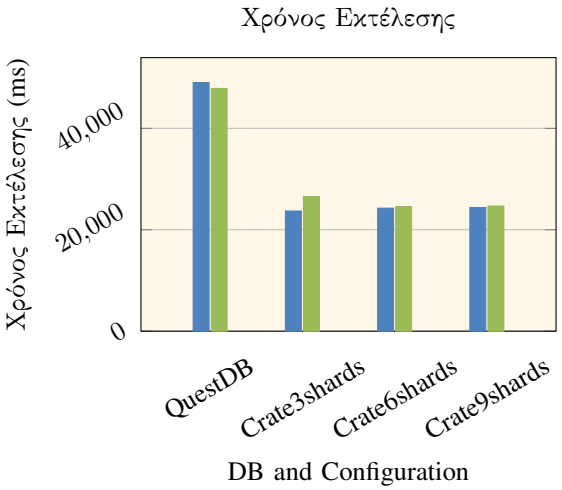
(f) Query Q6



(g) Query Q7



(h) Query Q8

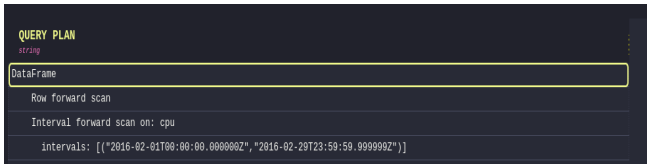


(i) Query Q9

Εικόνα 28: Χρόνοι Εκτέλεσης Custom Queries

(διότι το ASOF JOIN δημιουργεί matches με βάση το timestamp), δεν παρατηρείται το ίδιο και στην CrateDB, η οποία υλοποιεί την ίδια λειτουργία δημιουργώντας matches κάθε timestamp του πρώτου πίνακα με όλα τα μικρότερα timestamps του δεύτερου πίνακα και έπειτα κάνει ordering. Αυτή η μη αποδοτική υλοποίηση επιβαρύνεται από τη χρήση partitioning (η οποία τελικά αυξάνει τον συνολικό αριθμό των υπάρχοντων shards) καθώς τελικά επικρατούν τα overheads διαχείρισης και προσπέλασής όλων τους (η μη αποδοτική υλοποίηση οδηγεί σε προσπέλαση πολλών shards -και όχι μόνο των κοντινών χρονικά- και μετά κάνει ordering).

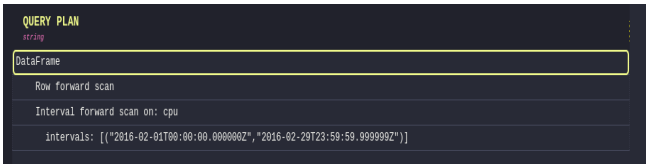
c) Query 3: Το query αυτό προσπελαύνει δεδομένα ενός μήνα, δηλαδή ενός partition. Ως εκ τούτου και οι δύο βάσεις δεδομένων επιτυγχάνουν καλύτερες επιδόσεις με χρήση partitioning, καθώς θα χρειαστεί να αναζητήσουν μόνο στα περίπου μισά rows (του ενός από τα δύο partitions), οδηγούμενες σε λίγότερο I/O χρόνο για διάβασμα από το δίσκο και λίγοτερο χρόνο αναζήτησης στα δεδομένα. Και πάλι, η QuestDB επιτυγχάνει καλύτερες επιδόσεις από την CrateDB. Το query plan για την questDB φαίνεται στην εικόνα 29. Για την εύρεση του χρονικού διαστήματος του ενός μήνα (που φαίνεται στην τελευταία γραμμή του plan), η questDB χρησιμοποεί forward interval scan, όπως αυτό περιεγράφηκε στο θεωρητικό τμήμα του project παραπάνω.



Εικόνα 29: Query 3 plan - Interval Scan

d) Query 4: Το query αυτό προσπελαύνει δεδομένα δύο μηνών, άρα δεδομένα και των δύο partitions και έτσι δεν απαλλάσσεται από την αναζήτηση ολόκληρου του dataset. Ως εκ τούτου, περιμένου το partitioning να μην επιδρά θετικά στην επίδοση, όπως και συμβαίνει. Μάλιστα στην CrateDB μπορεί και να οδηγεί σε ελαφρώς χειρότερες επιδόσεις, λόγω διαχειριστικού κόστους των πολλαπλών shards που δημιουργεί το partitioning. Συγκρίνοντας τις μετρήσεις με partitioning στα δύο queries 3 και 4, παρατηρούμε ότι στο query 3 (όπου τα partitions είναι χρήσιμα), η CrateDB μειώνει τον χρόνο εκτέλεσης θεαματικά, ενώ η QuestDB αν και μειώνει το χρόνο της, δεν το κάνει τόσο έντονα. Μπορούμε να συμπεράνουμε ενδεχομένως ότι η CrateDB εκμεταλλεύται τα partitions της πιο αποδοτικά συγκριτικά με την QuestDB. Ομοίως, το query plan για την questDB φαίνεται στην εικόνα 30, όπου φαίνεται η χρήση του interval scan.

e) Query 5: Στο σενάριο αυτό, η QuestDB τα πηγαίνει χειρότερα συγκριτικά με την CrateDB. Στο query αυτό, το κύριο χαρακτηριστικό ήταν η ύπραξη aggregations και groupings. Δεδομένου ότι αυτές οι πράξεις είναι αρκετά παραλληλοποιήσιμες, η CrateDB έχει καλύτερη επίδοση χάρη στο κατανεμημένο περιβάλλον της. Επιπλέον, η χρήση partitioning δεν βελτιώνει την επίδοση, καθώς αυτό το query διατρέχει όλο το σύνολο δεδομένων.



Εικόνα 30: Query 4 plan - Interval Scan

f) Query 6: Η QuestDB επιτυγχάνει αρκετά καλύτερες επιδόσεις από αυτές της CrateDB, καθώς το τριπλό JOIN (και γενικώς οι πράξεις με joins) απαιτεί -ανάλογα πάντα και με τον τρόπο υλοποίησής του- αρκετή επικοινωνία μεταξύ των κόμβων του κατανεμημένου περιβάλλοντος της CrateDB, κάτι που επιφέρει καθυστερήσεις. Το partitioning δεν βελτιώνει τις επιδόσεις καθώς το query διατρέχει ολόκληρο το σύνολο δεδομένων.

g) Query 7: Στο point query, αρχικά να σημειωθεί ότι η περίπτωση της QuestDB χωρίς partition (το οποίο σημαίνει εφαρμογή του default DAY partitioning) είναι εκτός συναγωνισμού καθώς προφανώς το day partitioning ευνοεί πολύ σημαντικά την εκτέλεση. Η QuestDB θα χρειαστεί να αναζητήσει μόνο εντός μια συγκεκριμένης μέρας (εντός ενός συγκεκριμένου partition) επιτυγχάνοντας πολύ καλή επίδοση. Εξαιρώντας αυτή την μη ενδειτική περίπτωση, η CrateDB επιτυγχάνει πολύ καλύτερες επιδόσεις στο point query και αυτό οφείλεται στο γεγονός ότι χρησιμοποιεί index σε όλα τα columns, όπως και στο timestamp, σε αντίθεση με την QuestDB (που δεν χρησιμοποιεί index στο timestamp).

h) Query 8: Στο GROUP BY/aggregation, η CrateDB έχει καλύτερη επίδοση από την QuestDB (έως και υποτριπλάσιος απαιτούμενος χρόνος σε κατάλληλο configuration). Οι πράξεις αυτές είναι εγγενώς παραλληλοποιήσιμες και ευνοούνται πολύ από το κατανεμημένο περιβάλλον της CrateDB αντί από το single node περιβάλλον της QuestDB. Μάλιστα, η ύπαρξη partitioning μπορεί να βελτιώσει την επίδοση καθώς οδηγεί σε αύξηση των υπάρχοντων shards, κάτι που αυξάνει τις δυνατότητες παραλληλισμού (καθώς στα shards μπορούν να εκτελούνται παράλληλα πράξεις). Βέβαια, όταν τα shards γίνονται υπεράριθμα, τα διαχειριστικά τους overheads επικρατούν (βλ. περίπτωση με 9 shards και partitioning).

i) Query 9: Το window function δεν επηρεάζεται από την χρήση ή μη partitioning, ενώ σταθερά φαίνεται να επιτυγχάνει αρκετά καλύτερες επιδόσεις στην CrateDB (σχεδόν υποδιπλάσιοι απαιτούμενοι χρόνοι). Οι συναρτήσεις αυτές που εμπεριέχουν grouping, ευνοούνται κατά την εκτέλεσή τους στο κατανεμημένο περιβάλλον της CrateDB.

Συνολικά, από τις παραπάνω μετρήσεις, εξάγεται ως συμπέρασμα ότι η QuestDB επιτυγχάνει καλύτερες επιδόσεις με χρήση ατομικών queries (όχι batches) με εξαίρεση τις περιπτώσεις όπου το query περιλαμβάνει aggregations, τα οποία φαίνεται ότι ευνοούνται από το κατανεμημένο περιβάλλον της CrateDB, ως αρκετά παραλληλοποιήσιμες πράξεις. Επιπλέον, αξιοσημείωτη είναι η υπεροχή της QuestDB σε

range queries (query 3 και 4) όπου γίνεται χρήση interval scan (ακόμη και στην περίπτωση του query 3 όταν η CrateDB χρησιμοποιεί partitioning). Η CrateDB σε αυτές τις περιπτώσεις χρησιμοποιεί το indexing (το οποίο ως γνωστόν εφαρμόζει σε όλα τα columns) για τον προσδιορισμό του time interval. Ωστόσο, το interval scan της QuestDB (η οποία δεν χρησιμοποιεί index στο timestamp) φαίνεται πιο αποδοτικό από το indexing της CrateDB.

VII. Ανακεφαλαίωση-Τελικά Συμπεράσματα

Κλείνοντας, συγκεντρώνουμε τα χαρακτηριστικά τελικά συμπεράσματα που παράχθηκαν από την παρούσα συγκριτική μελέτη των δύο βάσεων δεδομένων:

1) Ταχύτητα Data Ingestion:
   Στο data ingestion, η QuestDB επιτυγχάνει πάντοτε καλύτερες επιδόσεις (ειδικά για μεγάλα datasets) σε ό,τι αφορά τον απαιτούμενο χρόνο. Ωστόσο, ως προς την κλιμακωσιμότητα στο data ingestion (δηλαδή την ικανότητα να βελτιώνεται η επίδοση του data ingestion με αύξηση του πλήθους των ταυτόχρονων insertion requests - TSBS workers), η QuestDB είναι πολύ χειρότερη από την CrateDB, καθώς η ιδανική περίπτωση για αυτήν είναι η ατομική αποστολή insertion requests (1 TSBS Worker).
2) Memory consumption & compression:
   Το compression στην QuestDB φαίνεται να ενεργοποιείται και να γίνεται αποδοτικότερο όταν επιχειρείται το loading μεγαλύτερων datasets. Αντιθέτως η CrateDB εφαρμόζει αποδοτικά compression για κάθε σύνολο δεδομένων. Έτσι, ενώ σε μικρά datasets η CrateDB καταλαμβάνει πολύ λιγότερο χώρο (έως και 5 με 6 φορές λιγότερο χώρο), σε μεγάλα datasets, ο χώρος που καταλαμβάνεται από τις δύο βάσεις γίνεται περίπου ίσος. Το προφανές πλεονέκτημα της CrateDB βέβαια είναι ότι αξιοποιεί τα συστήματα μνήμης πολλαπλών κόμβων έχοντας έτσι μεγαλύτερη άνεση στην αποθήκευση μεγαλύτερων datasets συγκριτικά με την single node QuestDB, αλλά με μεγαλύτερο κόστος.
3) Εκτέλεση batches από queries:
   Η CrateDB επιτυγχάνει καλύτερες επιδόσεις όταν λαμβάνει παράλληλα αιτήματα για εκτέλεση queries καθώς το κατανεμημένο σύστημά της ευνοεί την παραλληλία στην ικανοποίησή τους. Επιπλέον η CrateDB χαρακτηρίζεται από καλύτερη κλιμακωσιμότητα, δηλαδή ικανότητα βελτίωσης της επίδοσης με αύξηση των παράλληλων requests για εκτέλεση queries. Η QuestDB επίσης διατηρεί cache με τα αποτελέσματα προηγούμενων queries (κάτι που φάνηκε από την εντυπωσιακή μείωση του χρόνου εκτέλεσης ενός query όταν ένα παρόμοιο query εκτελέστηκε προηγουμένως - αυτό δεν συναντάται στην CrateDB). Σε αντίθεση με την CrateDB, η QuestDB -ως single node σύστημα- δεν καταφέρνει να διαχειριστεί αποδοτικά την παράλληλη ικανοποίηση πολλαπλών requests -ακόμη και όταν αυτά αφορούν την εκτέλεση του ίδιου είδους query, κάτι που θα μπορούσε να ευνοηθεί από τη διατήρηση cache αποτελεσμάτων προηγούμενων queries. Όταν στα batches από queries χρησιμοποιείται ένας TSBS worker (δηλαδή τα requests στέλνονται ατομικά), η QuestDB επιτυγχάνει μόνο λίγο χειρότερες επιδόσεις από αυτές της CrateDB κάτι που -όπως θα φανεί στη συνέχεια- αναδεικνύει την εν γένει (με εξαιρέσεις) πλεονεκτική θέση της QuestDB στην ικανοποίηση ατομικών requests για εκτέλεση queries.
4) Εκτέλεση ατομικών (custom) queries:
   Στην ικανοποίηση ατομικών αιτημάτων για εκτέλεση queries, η QuestDB για τα περισσότερα είδη από queries επιτυγχάνει καλύτερες (έως και εντυπωσιακά καλύτερες) επιδόσεις. Εξαίρεση αποτελούν τα point queries και τα aggregations. Τα πρώτα ευνοούνται από το indexing της CrateDB (το οποίο εφαρμόζεται σε όλα τα columns, σε αντίθεση με την QuestDB στην οποία εφαρμόζεται μόνο σε columns τύπου Symbol). Τα δεύτερα -ως εγγενώς αρκετά παραλληλοποιήσιμες πράξεις- ευνοούνται από το κατανεμημένο περιβάλλον της CrateDB. Στα range queries, το interval scan της QuestDB υπερισχύει του indexing της CrateDB.
   Τελικά: η χρήση καμίας από τις δύο βάσεις δεν αποτελεί πανάκεια για την διαχείριση δεδομένων πραγματικού χρόνου. Για παράδειγμα, εφαρμογές αυτού του είδους είθισται να υποβάλλουν ερωτήματα που περιέχουν aggregations και ερωτήματα που περιλαμβάνουν conditions σε κάποια time intervals (range queries), με την CrateDB να υπερισχύει στα πρώτα και την QuestDB στα δεύτερα. Ανάλογα με τις απαιτήσεις της προς υλοποίηση εφαρμογής (π.χ. απαίτηση για μικρό memory consumption, για παράλληλη εκτέλεση πολλών requests, για ταχύτερη αποθήκευση δεδομένων πραγματικού χρόνου κ.λπ.), ο σχεδιαστής καλείται να επιλέξει μεταξύ των δύο βάσεων δεδομένων αναλογιζόμενος τα ανωτέρω ζητήματα που θίχτηκαν στο παρόν project.

REFERENCES

[1] TSBS Github
https://github.com/timescale/tsbs

[2] QuestDB webpage: Quick start. This guide will get your first QuestDB instance running
https://questdb.io/docs/quick-start
/#default-root-directory

[3] CrateDB webpage: Getting started, Installing and Configuring
https://cratedb.com/docs/crate/tutorials/en/latest/basic/index.html
#debian-or-ubuntu

[4] CrateDB multi-node setup
https://cratedb.com/docs/crate/howtos/en/latest/clustering
/multi-node-setup.html

[5] Go webpage : Download and install
https://go.dev/doc/install

[6] SSH Tunneling
https://www.ssh.com/academy/ssh/tunneling

[7] QuestDB: Introduction
https://questdb.io/docs

[8] QuestDB: Configuration
https://questdb.io/docs/configuration/

[9] QuestDB: Data Insertion
https://questdb.io/docs/develop/insert-data/

[10] QuestDB: Data Deduplication
https://questdb.io/docs/concept/deduplication/

[11] QuestDB: Write-Ahead Log
https://questdb.io/docs/concept/write-ahead-log/

[12] QuestDB: Storage Model
https://questdb.io/docs/concept/storage-model/

[13] QuestDB: Partitions
https://questdb.io/docs/concept/partitions/

[14] QuestDB: Τύπος Symbol
https://questdb.io/docs/concept/symbol/

[15] QuestDB: Indexing
https://questdb.io/docs/concept/indexes/

[16] QuestDB: Interval Scan
https://questdb.io/docs/concept/interval-scan/

[17] QuestDB: Write Amplification
https://questdb.io/docs/deployment/capacity-planning/#write-amplification

[18] QuestDB: High Cardinality
https://questdb.io/glossary/high-cardinality

[19] Write Amplification
https://www.tuxera.com/blog/what-is-write-amplification
-why-is-it-bad-what-causes-it/

[20] CrateDB in general
https://cratedb.com/product

[21] CrateDB: Lucene Engine
https://cratedb.com/product/features/lucene-engine

[22] CrateDB: Columnar Storage % Indexing
https://cratedb.com/product/features/indexing-columnar-storage-aggregations

[23] CrateDB: Shared-Nothing Architecture
https://cratedb.com/product/features/shared-nothing-architecture

[24] CrateDB: Dynamic Database Schemas
https://cratedb.com/product/features/dynamic-schemas

[25] CrateDB: High Availability
https://cratedb.com/product/features/high-availability

[26] CrateDB: Full-text Search
https://cratedb.com/product/features/full-text-search

[27] CrateDB Workshop playlist
https://www.youtube.com/playlist?list=
PLDZqzXOGoWUIm2Jsvs5cjpBACe-YRamk7

[28] CrateDB: Time-series data
From raw data to fast analysis in only three steps
https://youtu.be/7biXPnG7dY4

[29] CrateDB:Data Replication
https://cratedb.com/product/features/data-replication

[30] CrateDB:Partitioning
https://cratedb.com/product/features/partitioning

[31] CrateDB:Sharding
https://cratedb.com/product/features/sharding

[32] CrateDB:Optimization
https://cratedb.com/docs/crate/reference/en/5.6/admin
/optimization.html