PERF UG

Programmation réactive

Février 2014

There is a better way

OCTO Technology

Sommaire

Contexte

Contexte

- Recevoir en **temps réel** des informations de capteur à haute fréquence
- Effectuer des calculs et **enrichir** ces informations en **temps réel**
- **Stocker** ces informations de manière sûre
- Distribuer en **temps réel** à un nombre important d'utilisateur
- Être **tolérant** à la panne

Constat

- L’application existante ne pourra pas absorber le volume futur
- Le nombre de client été décuplé en 5 ans
  x 50
- Performance / exploitant

Problématiques

Les différentes problématiques sous jacentes sont:

- Injection **haute fréquence**
- Temps réel
- Persistance **haute fréquence**
- Haute disponibilité

Exigences fonctionnelles

- Couvrir le périmètre existant
  Sans calculs
- Pas d'ordre sur l'arrivée des messages
- Pas d'ordre sur la distribution
- La distribution prime sur le stockage

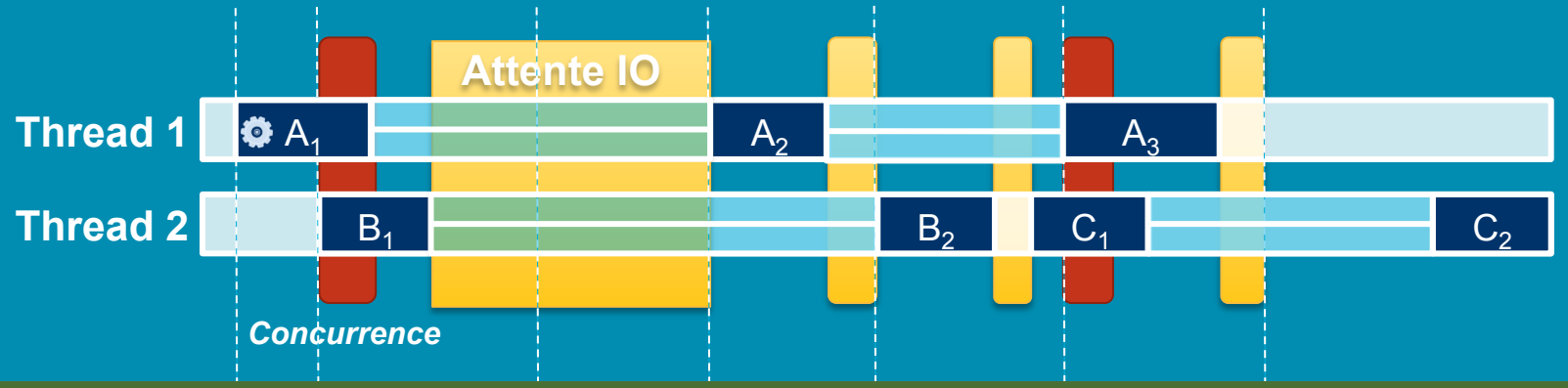
Reactive ?

There is a better way

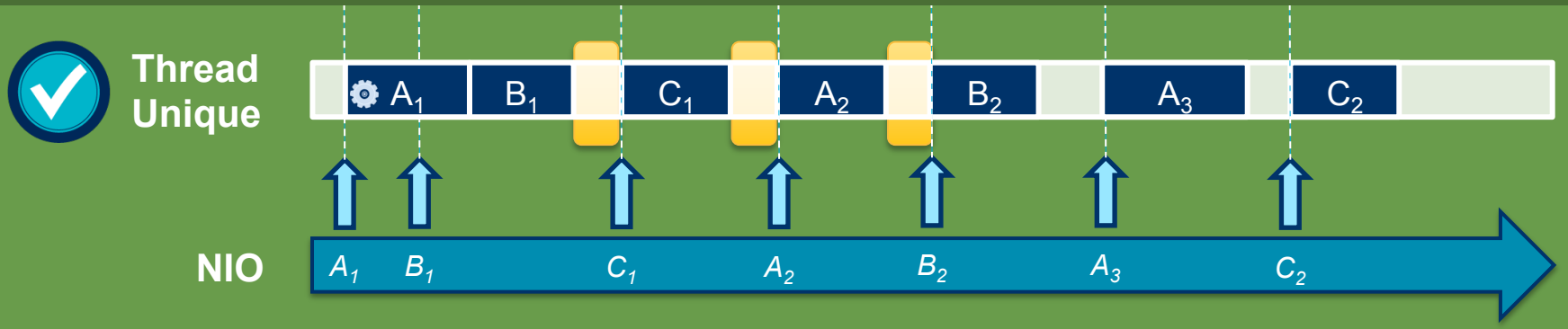
OCTO Technology

Concurrence des traitements

Approche « probabiliste » (actuelle)



Approche Réactive (Optimise la CPU)



Calcul

Attente I/O

Enjeux

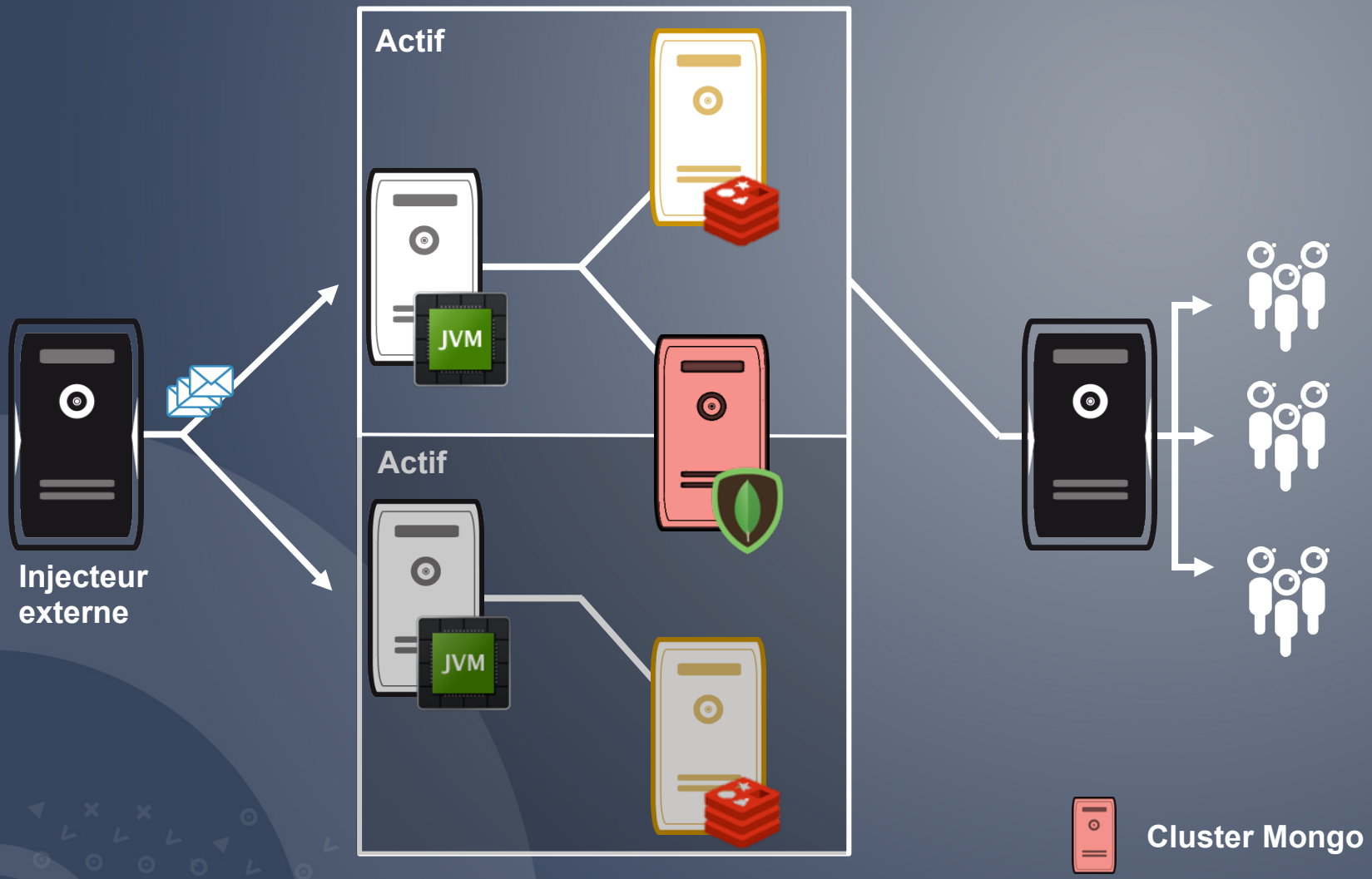
- Approche **réactive** vs approche **classique**
- **SQL** vs **NoSQL** sur la même architecture
- No **Scala** Fanboy !

Architectures

There is a better way

OCTO Technology

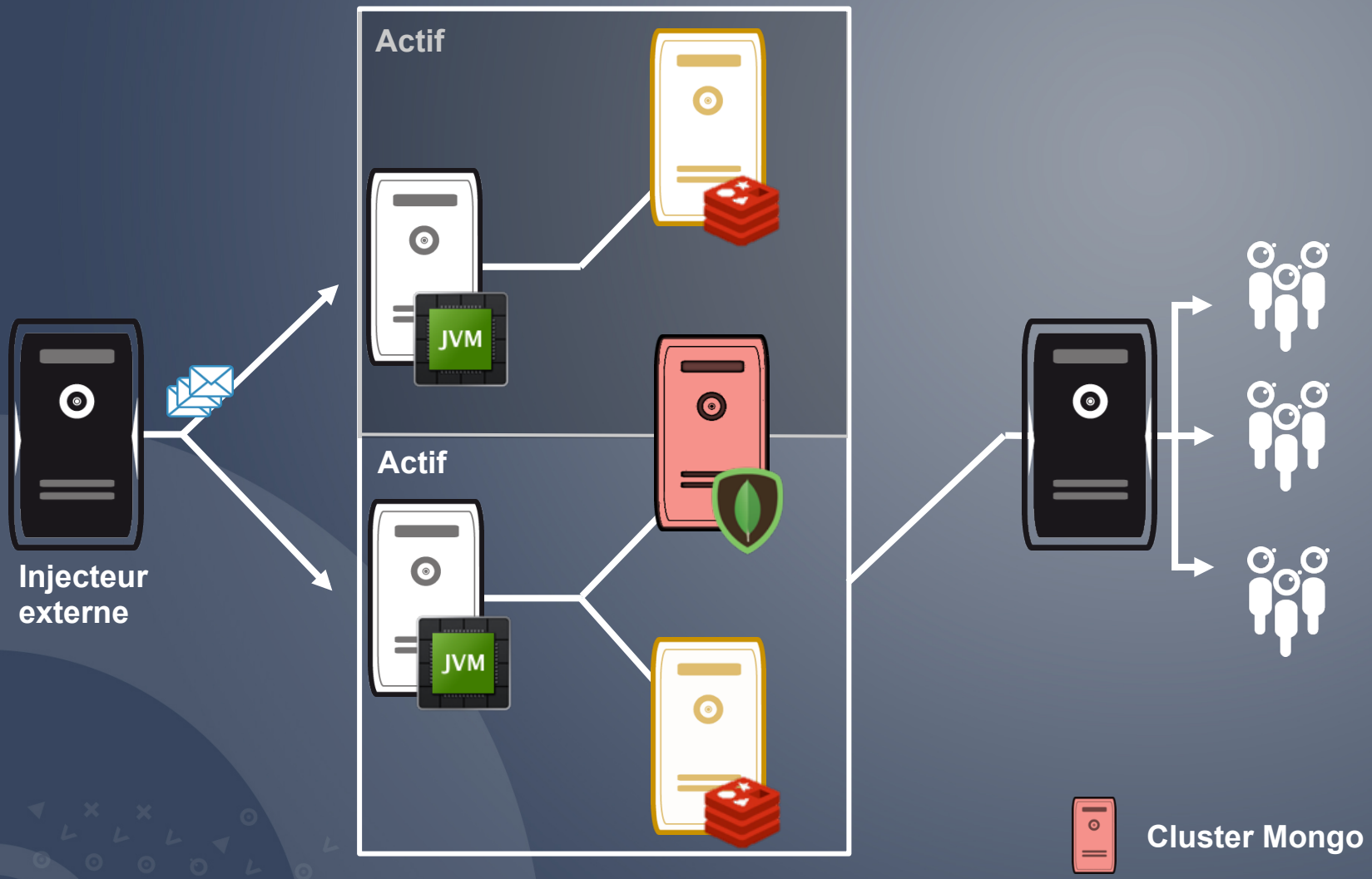
Architecture Haute Dispo



There is a better way

OCTO Technology

Architecture Haute Dispo



There is a better way

OCTO Technology

Stockage

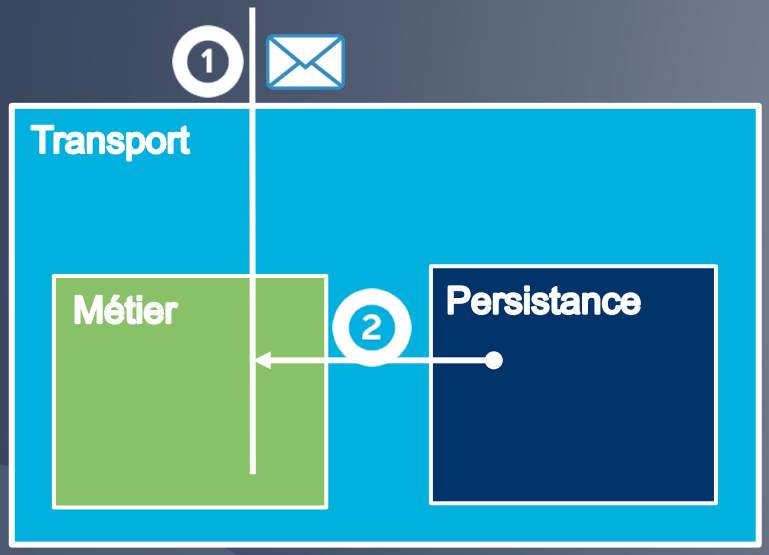
Type	Durée	Usage	Sizing
RAM	2h – 1j	Utilisé en live (cache)	< 500 Mo
Redis, non persisté	2h – 1j	Utilisé en cas de fail (backup de la ram)	< 500 Mo
Mongo / MySQL	24 h / 7j 1 message / 5	Stockage à froid Requête historique (> 24h)	Non déterminé *

* Tirs 4Go de données au format CSV

Architecture Fonctionnelle

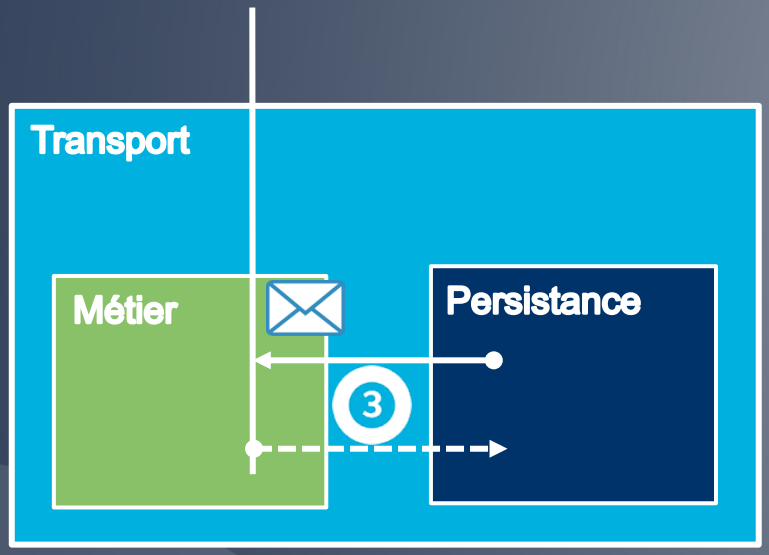
1. Transport
   Reçoit et diffuse les messages
2. Métier
   Traite et enrichit les messages
3. Persistance
   Sauvegarde en cascade les messages

Architecture Fonctionnelle



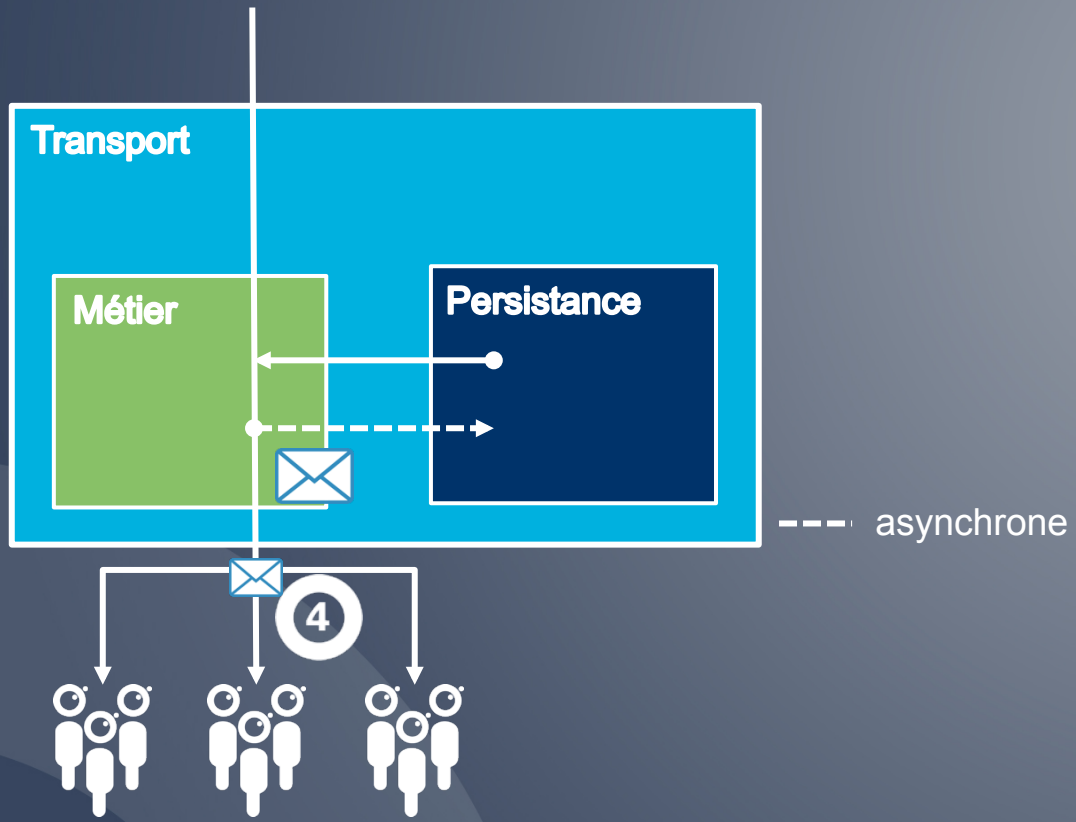
1. Réception d'un message
2. Enrichissement du message (1 select)

Architecture Fonctionnelle



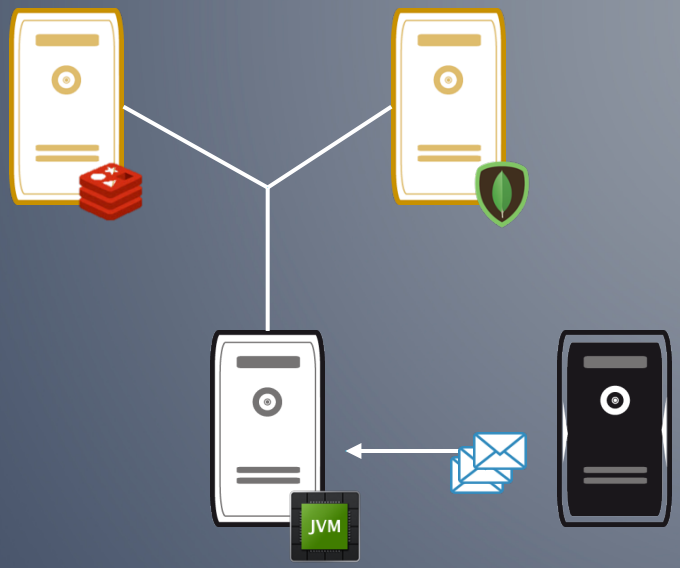
3 Persistance
Sauvegarde en cascade les Messages
1 insert – 1 commit

Architecture Fonctionnelle



4 Diffusion
via web socket aux clients

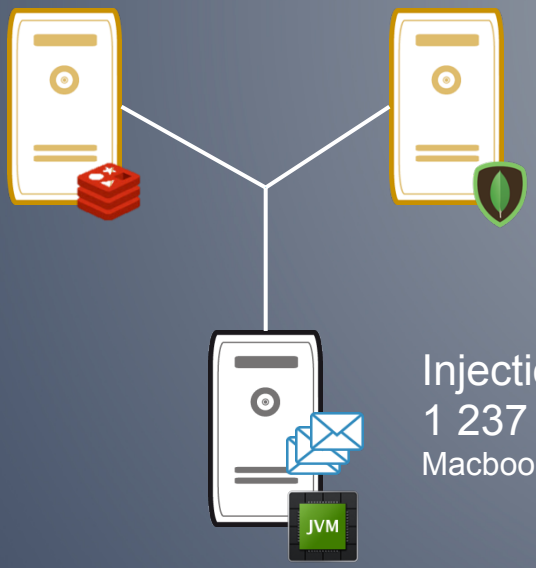
Architecture Technique



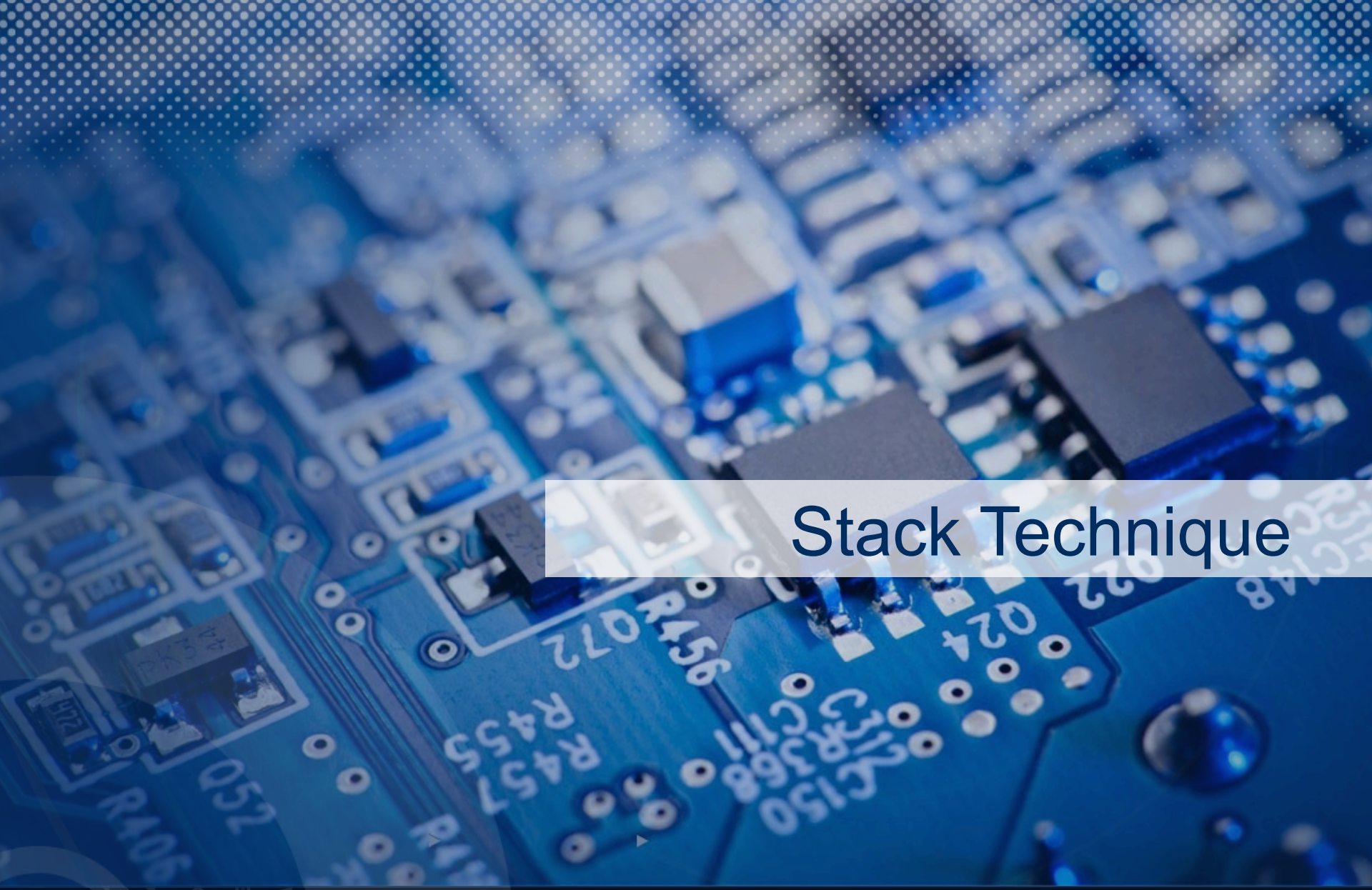
There is a better way

OCTO Technology

Architecture Technique



Injection intra jvm à l'aide de **MMAP**
1 237 623 messages / seconde
Macbook pro i7



Stack Technique

Stack Technique

Langages

- Scala
- Akka

Perf Test

- Akka Remote
- Async http client
- ~~Gatling / JMeter~~

Unit Tests

- ScalaTest
- EasyMockSugar
- Jacoco & Scct

Tuning Tools

- Yourkit
- jClarity
- Censum

Choix des Drivers

MySQL:

- **Sync** : mysql.mysql-connector-java
- **Async** : org.adbcj.mysql-async-driver

Mongo:

- **Sync** : org.mongodb.casbah
- **Async** : org.reactivemongo reactivemongo

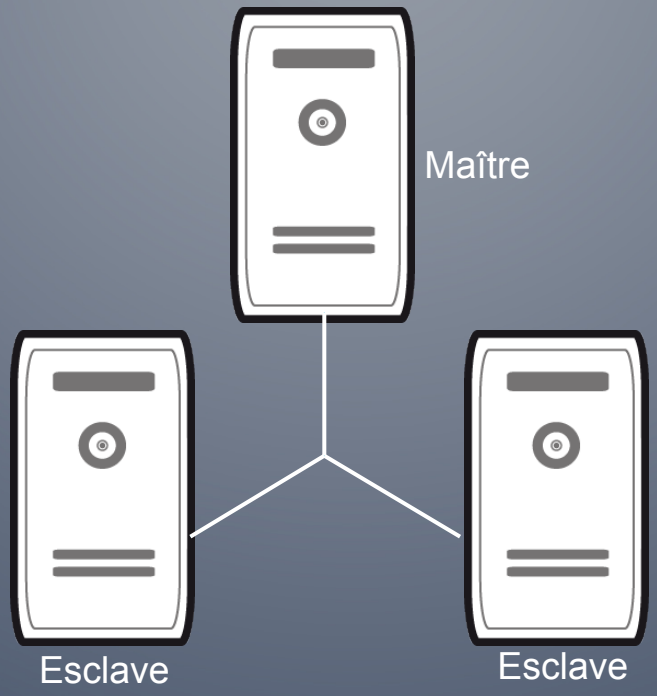
Redis:

- **Sync** : net.debasishg.redisclient
- **Async** : com.etaty.rediscala rediscala

Architecture Technique

amazon web services™ | EC2

Type	**C1 xLarge**
Architecture	**64 bits**
Nombre de coeurs	**8**
RAM	**7 Go**
Réseau	**Elevé**
HDD	**Ephemeral**
Distribution	**Ubuntu 13.10**



Architecture des Tirs

There is a better way

OCTO Technology

Architecture de Test

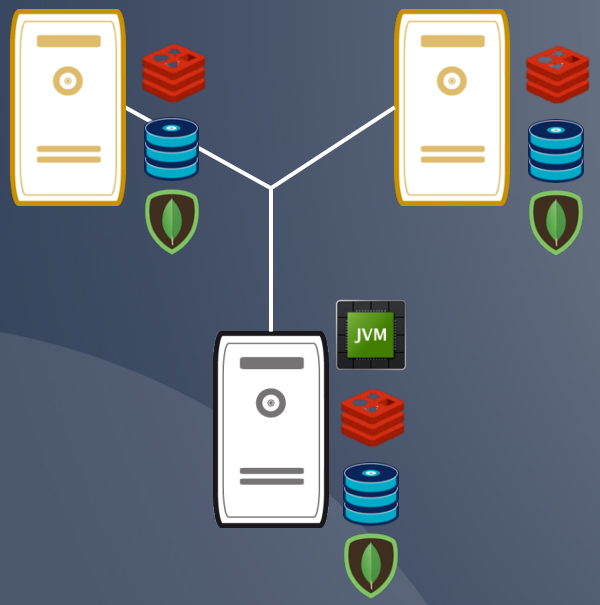
Nœuds Esclaves

Nœud Maître

There is a better way

OCTO Technology

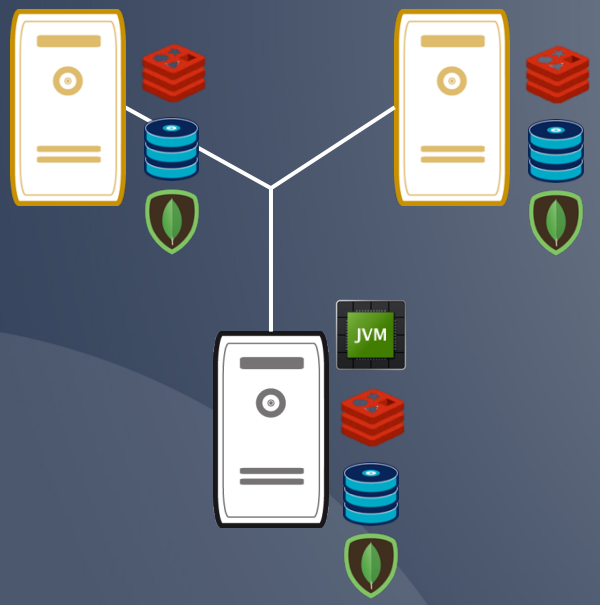
Exemple de configuration



Activation de la brique technique par configuration

Plus de souplesse

Exemple de configuration



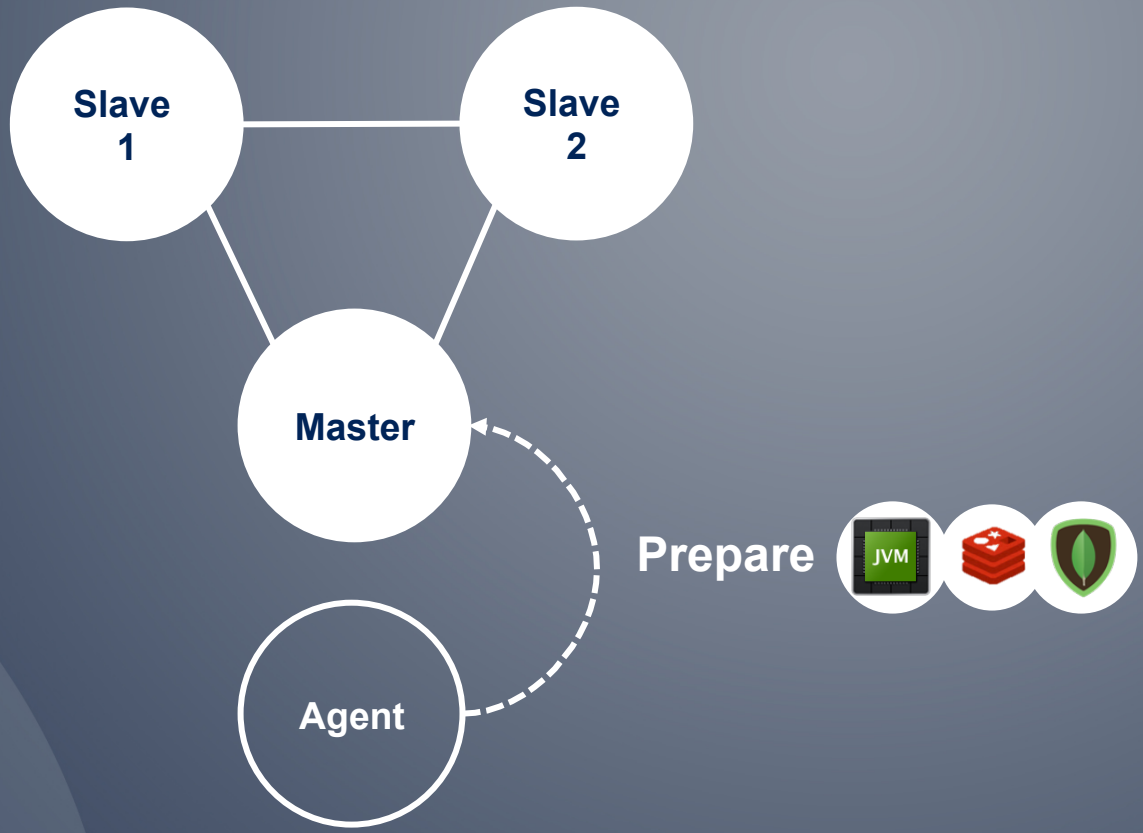
Activation de la brique technique par configuration

Plus de souplesse

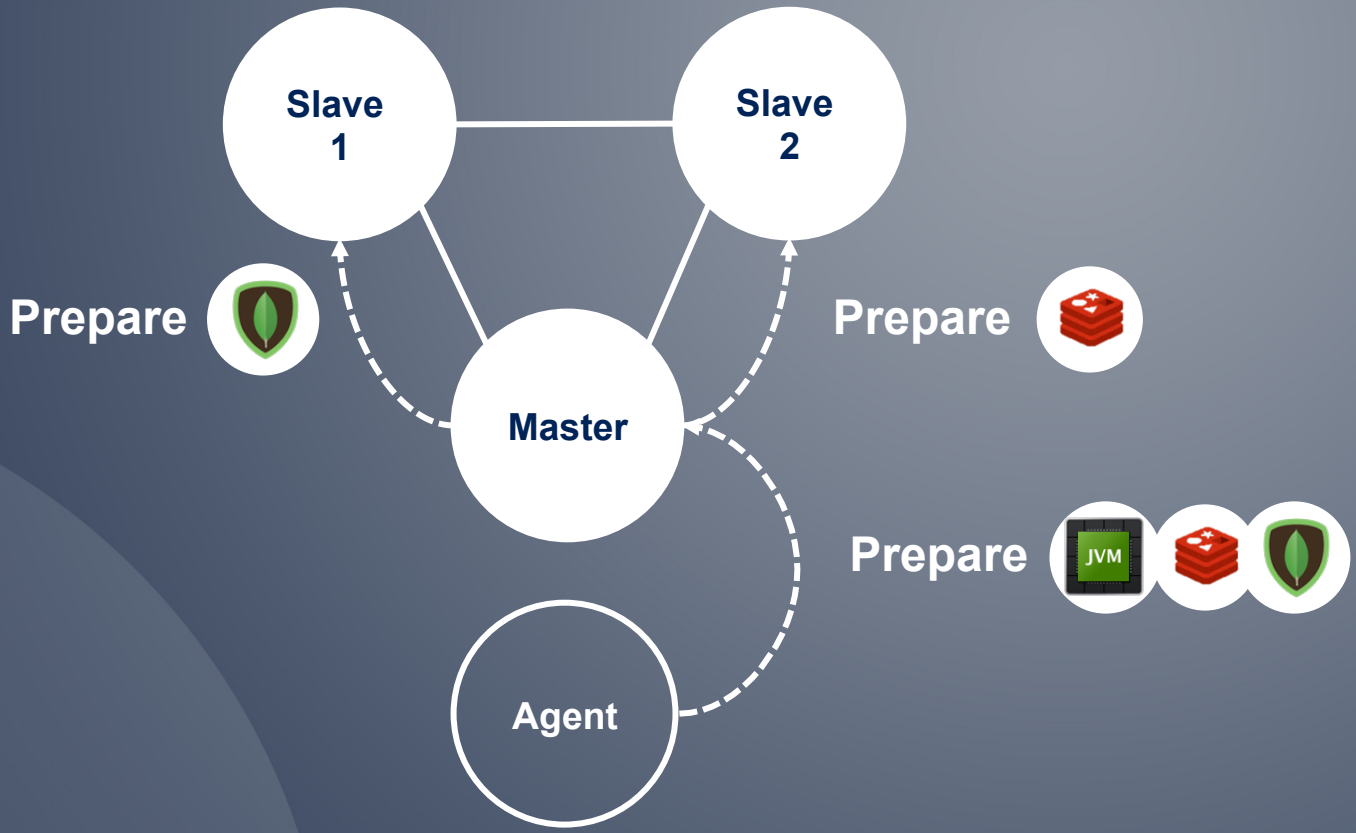
There is a better way

OCTO Technology

Akka Remote 1/4



Akka Remote 2/4



There is a better way

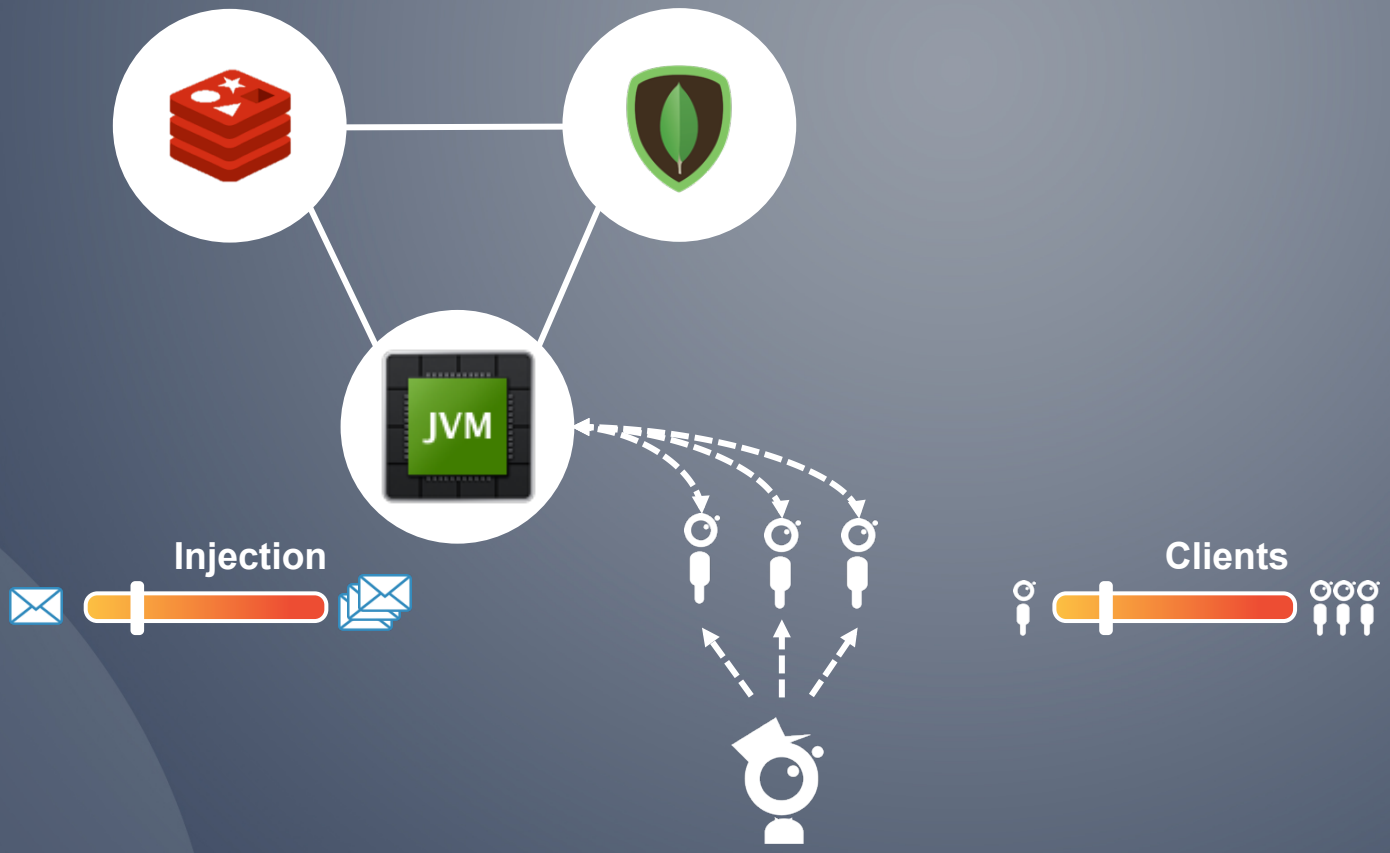
OCTO Technology

Akka Remote 3/4

There is a better way

OCTO Technology

Akka Remote 4/4



Protocol de test

There is a better way

OCTO Technology

Combinaisons testées

Sans persistance
Avec et sans calculs

STM
Software Transactionnal Memory

Lock
Approche pessimiste

Mysql
async / sync

Redis
async / sync

Mongo
async / sync

Redis async,
Mongo sync

Redis async,
Mongo async

Redis async,
Mysql sync

Redis async,
Mysql async

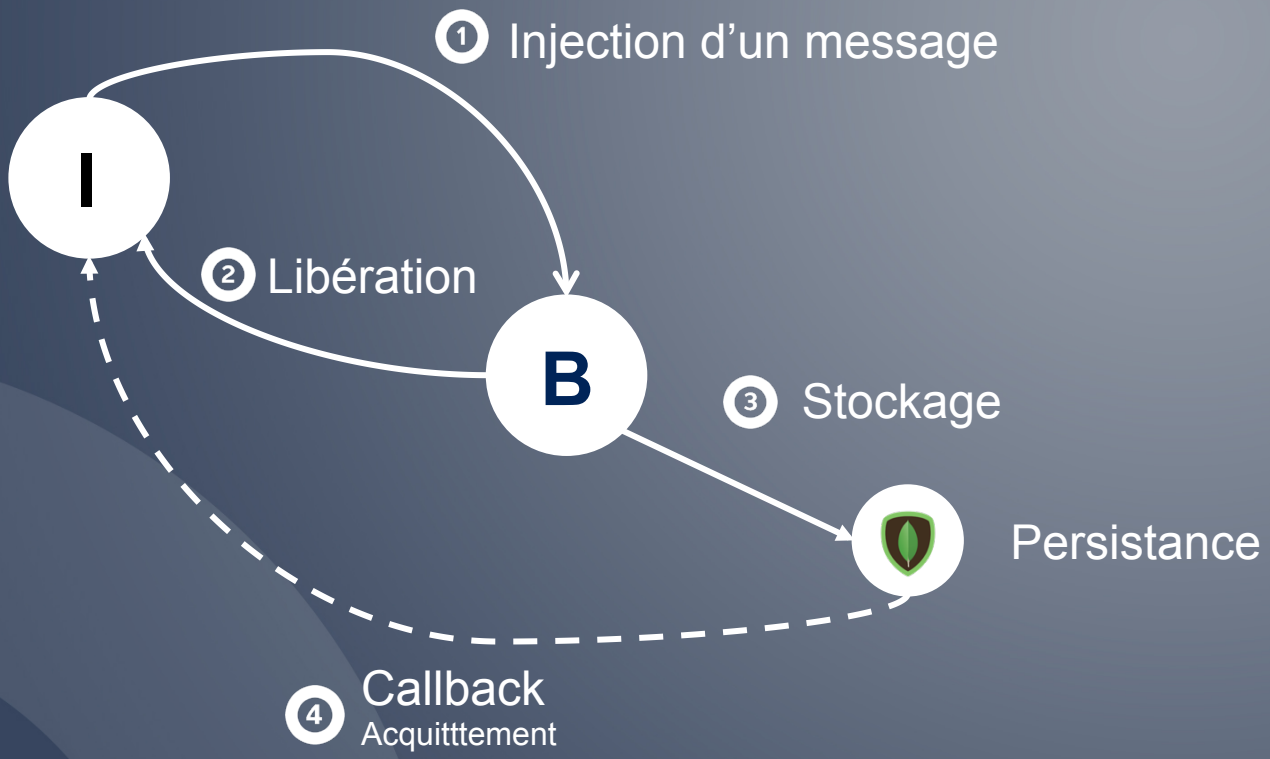
There is a better way

OCTO Technology

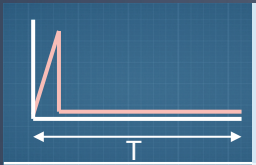
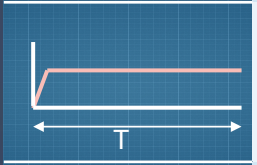
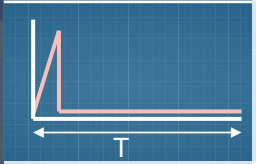
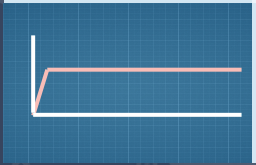
Bottlenecks

- Warning: Un système réactif ne se teste pas comme un système synchrone
- Les bottlenecks d'un système asynchrone ne sont pas les mêmes que ceux d'un système synchrone

Injecteur



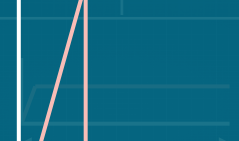
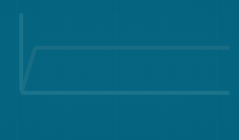
Scénarios de test

Type	Etat initial	Caractéristiques	Nombre de client
	Vide	T = 1s, 5s et 10s	1 client RT
	Vide	T = 1s 80 %, 90 %, 100 % du peak	1 client RT
	Vide	T = 1s	100, 200, 1000 clients RT
-	10 000 records	-	100, 200, 500 search queries (10 000 records)
	Vide	80% du peak 1 an de message (27 M)	1 client RT

There is a better way

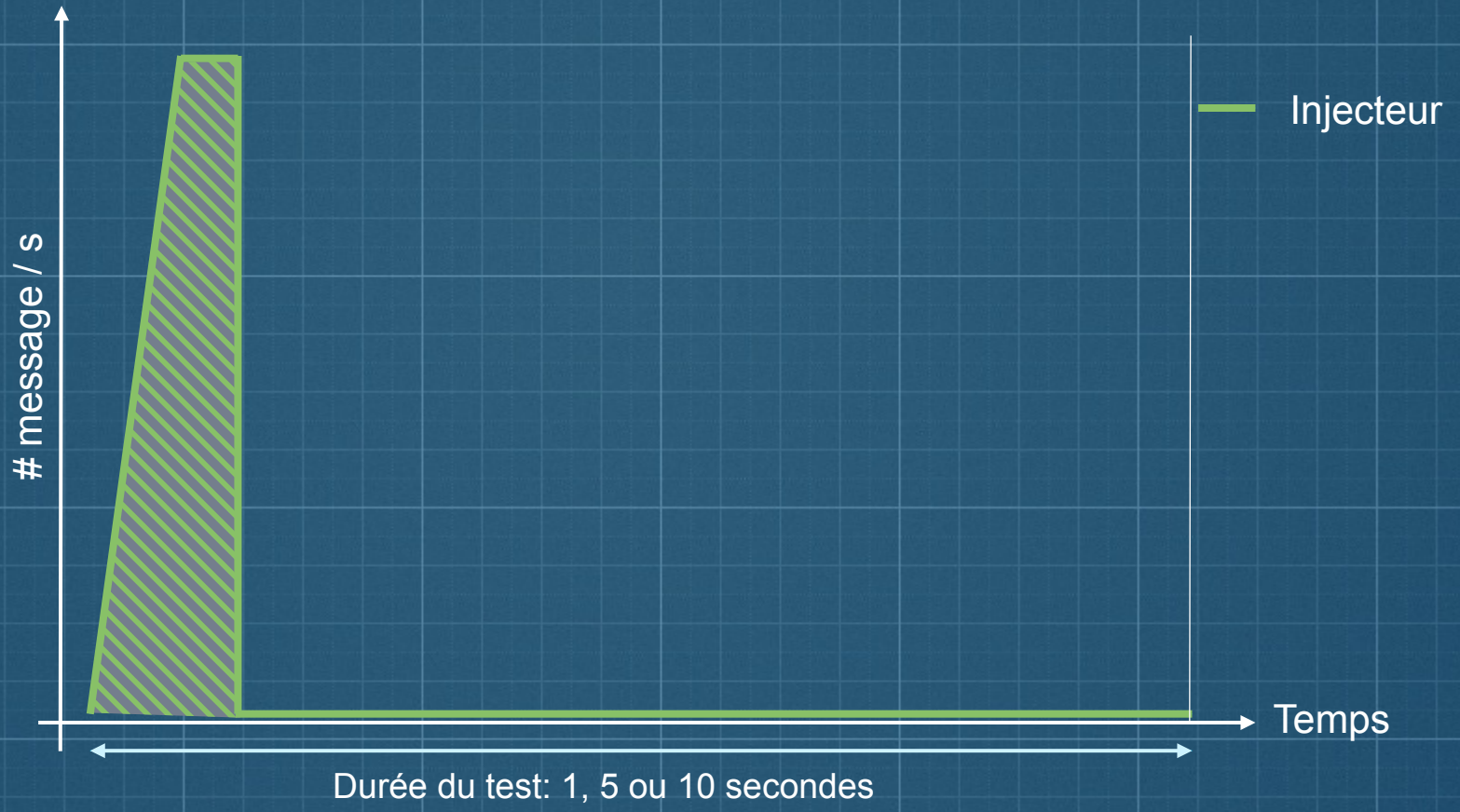
OCTO Technology

Scénarios de test

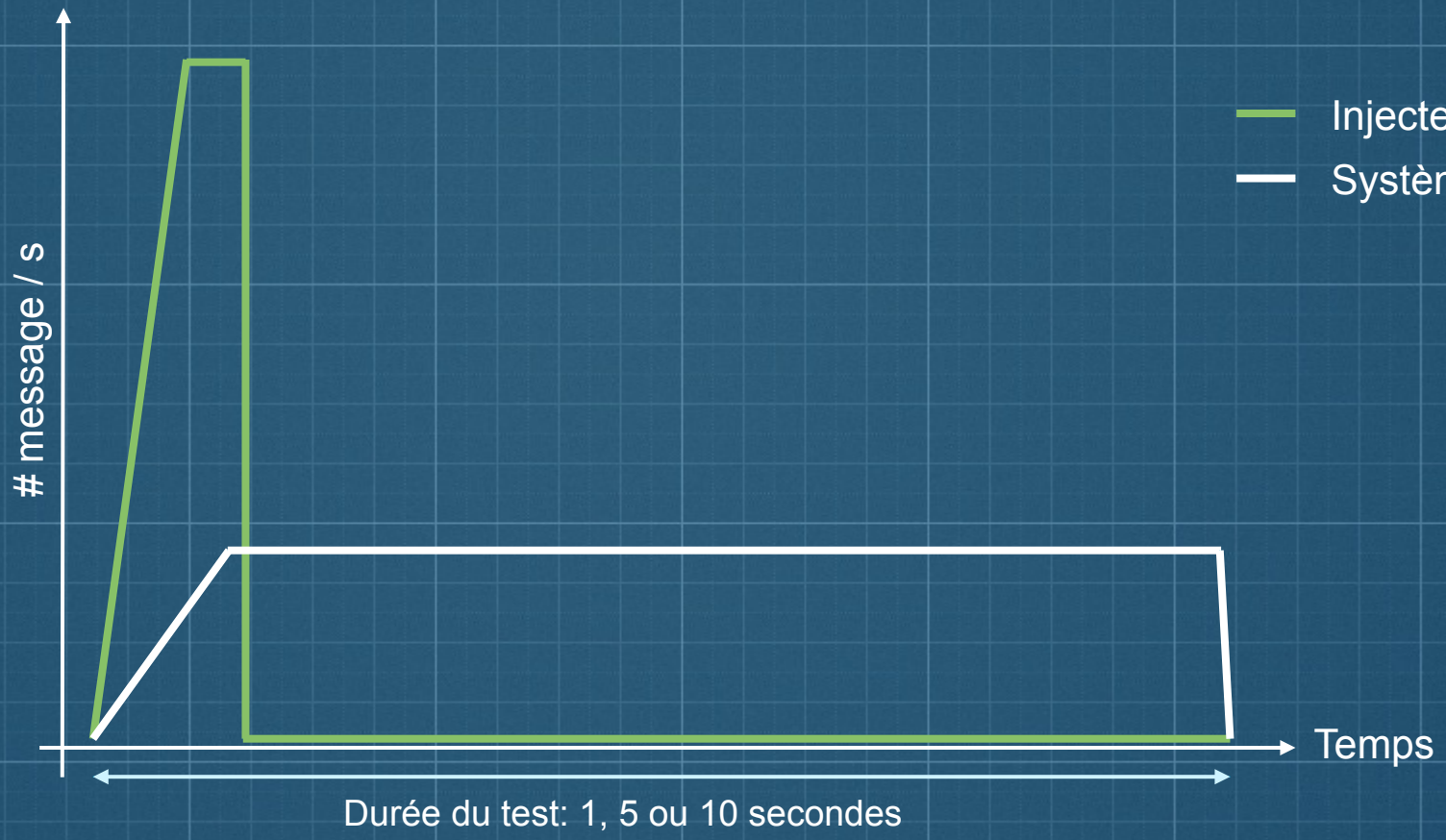
Type	Etat initial	Caractéristiques	Nombre de client
	Vide	T = 1s, 5s et 10s	1 client RT
	Vide	80 %, 90 %, 100 % du peak	1 client RT
	Vide	T = 1s	100, 200, 1000 clients RT
-	10 000 records	-	100, 200, 500 search queries (10 000 records)
	Vide	80% du peak 1 an de message (27 M)	1 client RT

Débit auto-ajusté

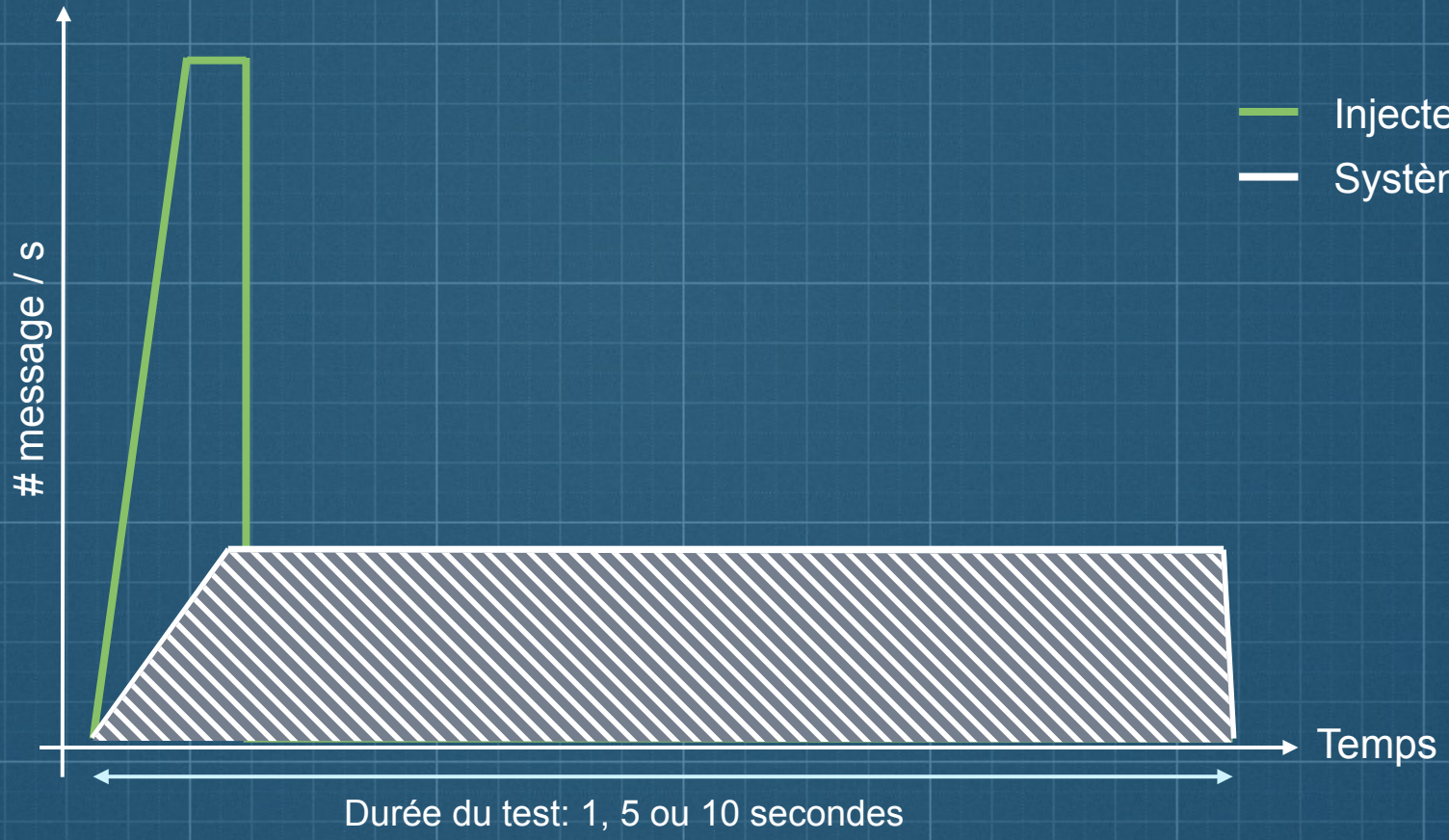
Injection auto ajustée



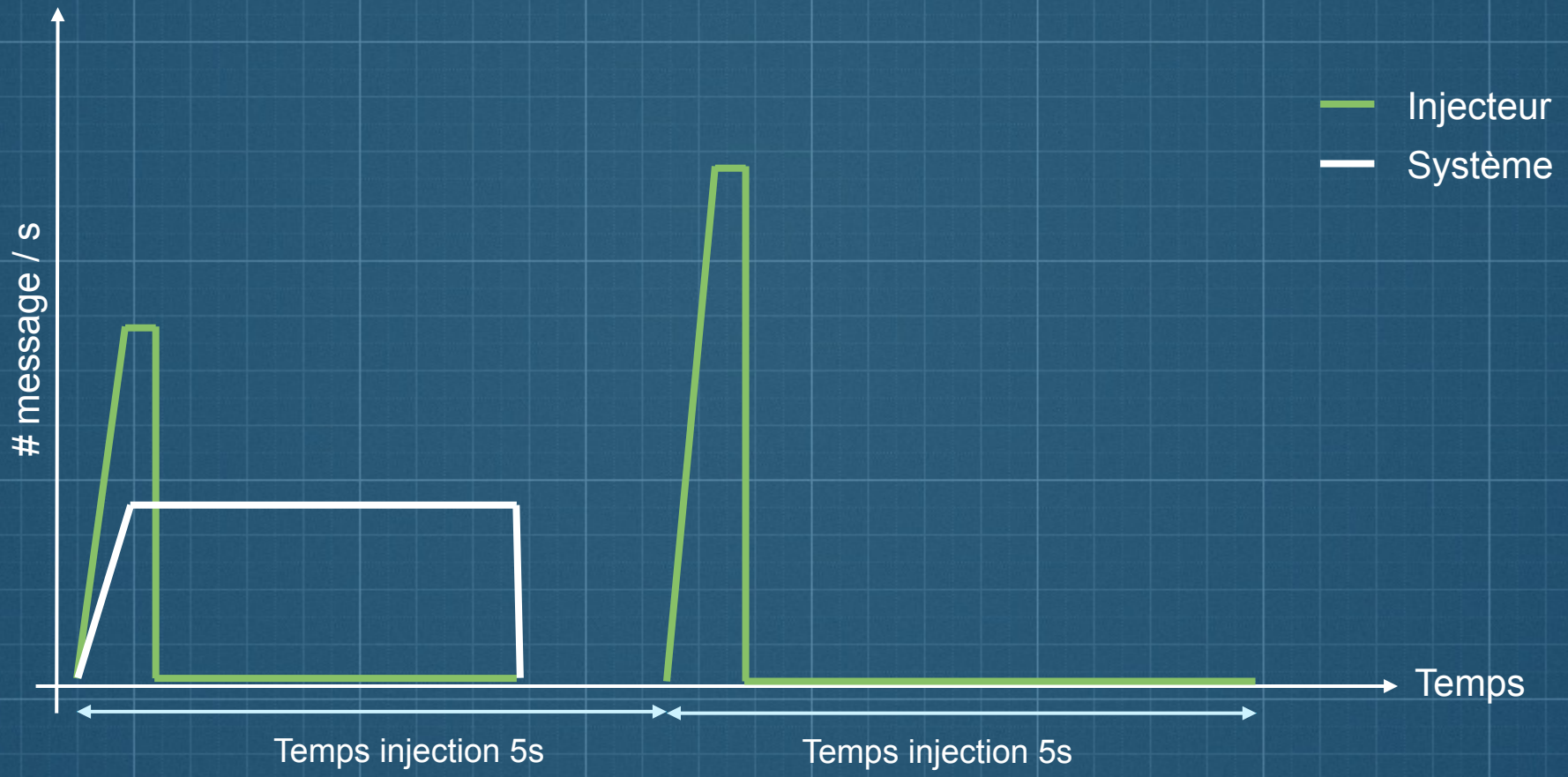
Injection auto ajustée



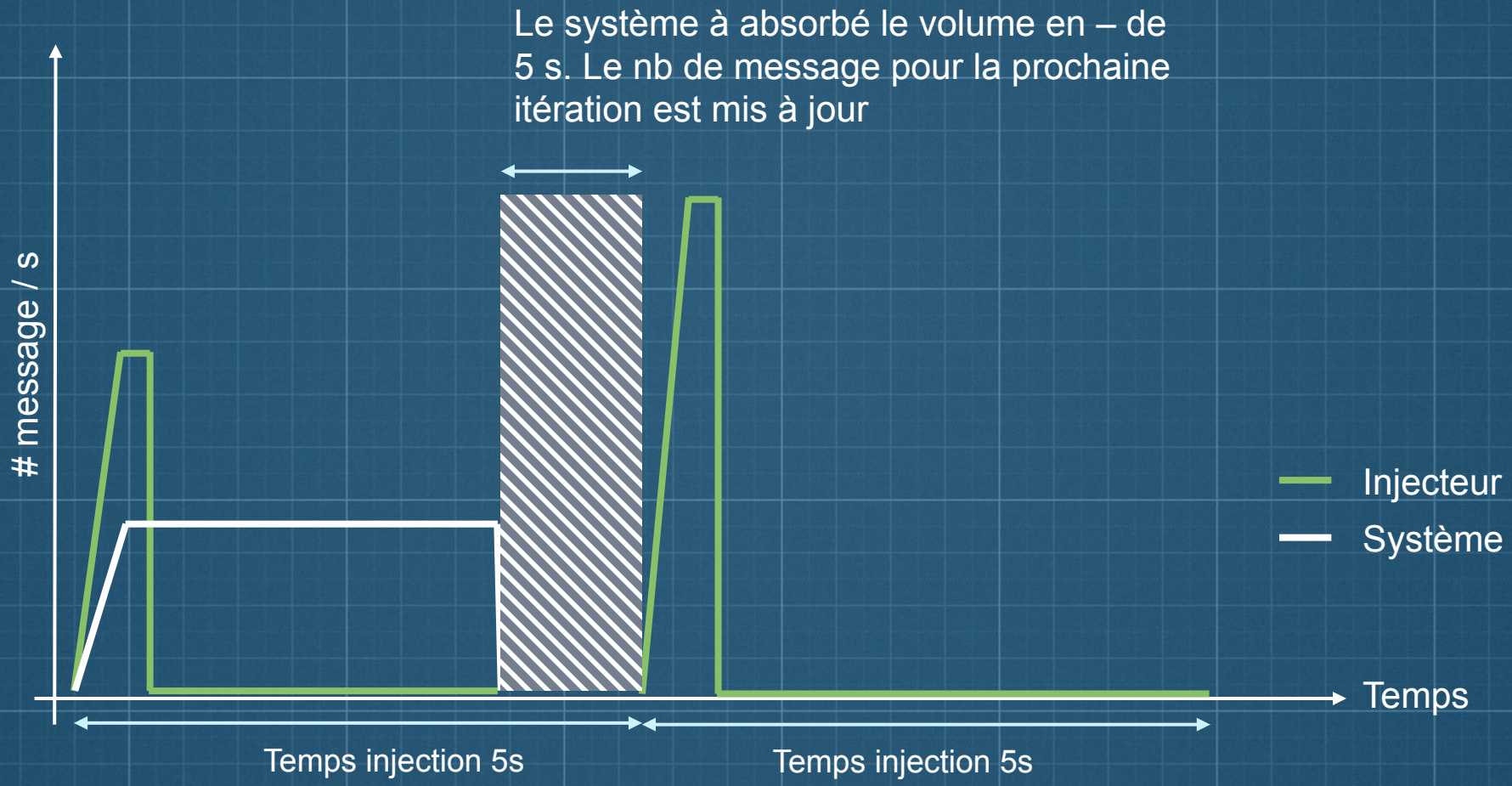
Injection auto ajustée



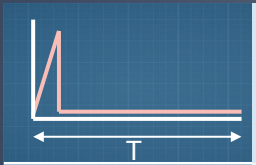
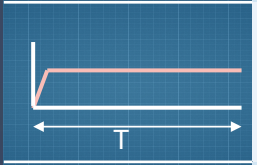
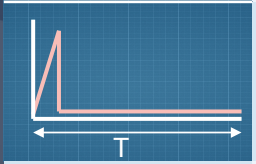
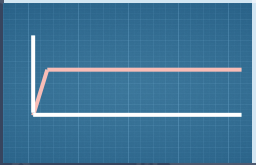
Injection auto ajustée



Injection auto ajustée



Scénarios de test

Type	Etat initial	Caractéristiques	Nombre de client
	Vide	T = 1s, 5s et 10s	1 client RT
	Vide	T = 1s 80 %, 90 %, 100 % du peak	1 client RT
	Vide	T = 1s	100, 200, 1000 clients RT
-	10 000 records	-	100, 200, 500 search queries (10 000 records)
	Vide	80% du peak 1 an de message (27 M)	1 client RT

There is a better way

OCTO Technology

En Chiffre

There is a better way

OCTO Technology

Performance
MySQL

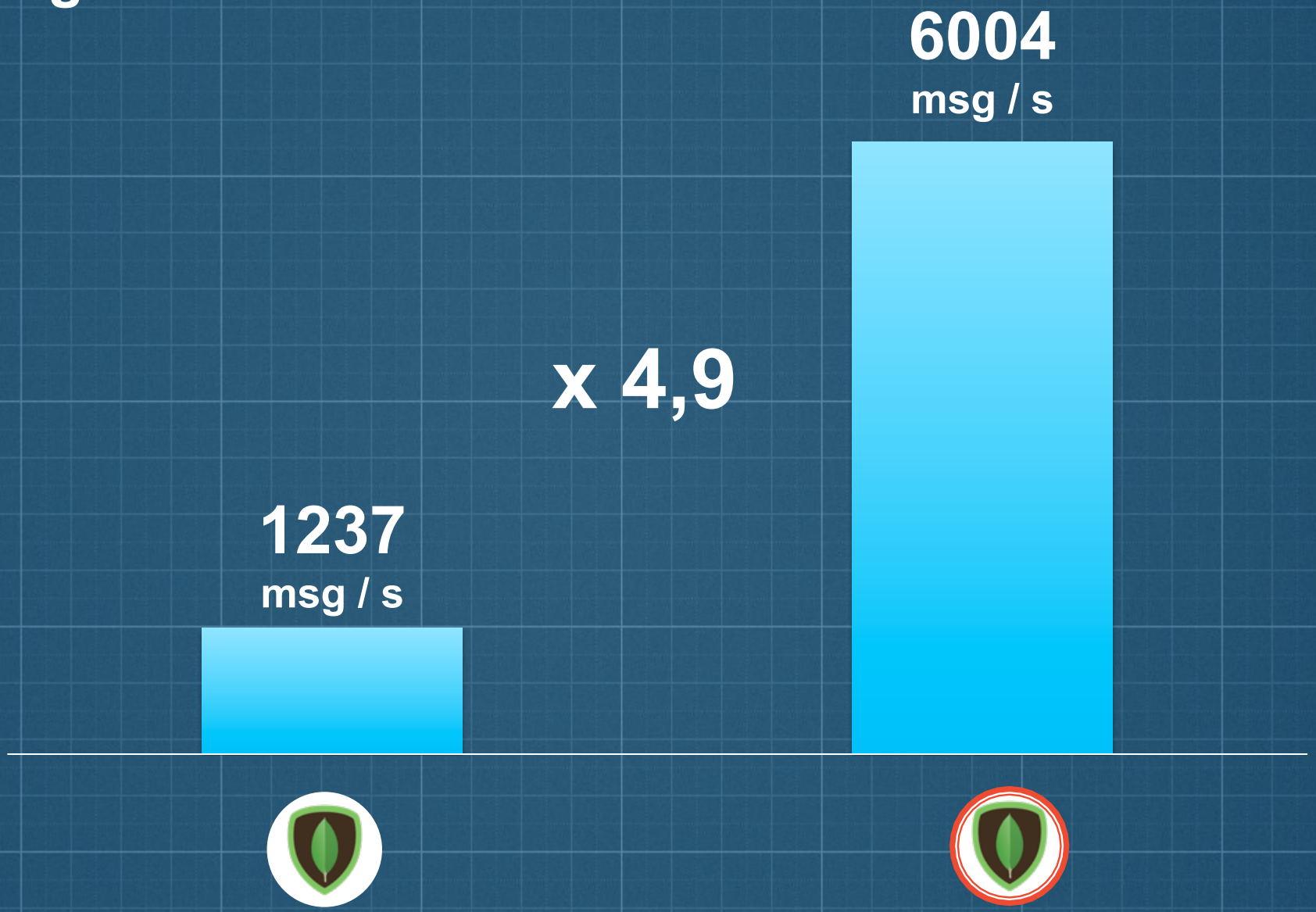
VS

Performance
MySQL

Performance
Mongo

VS

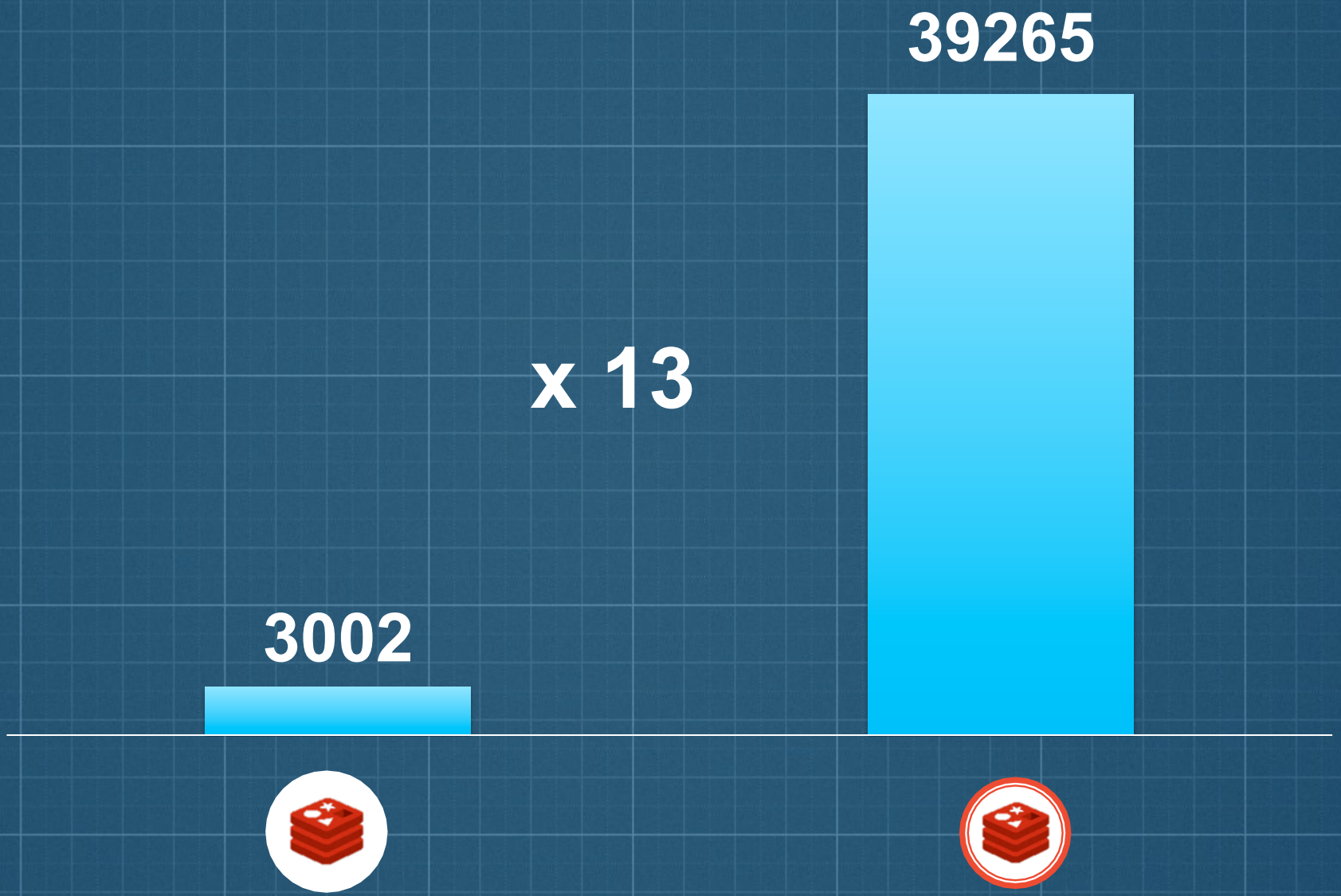
Performance
Mongo



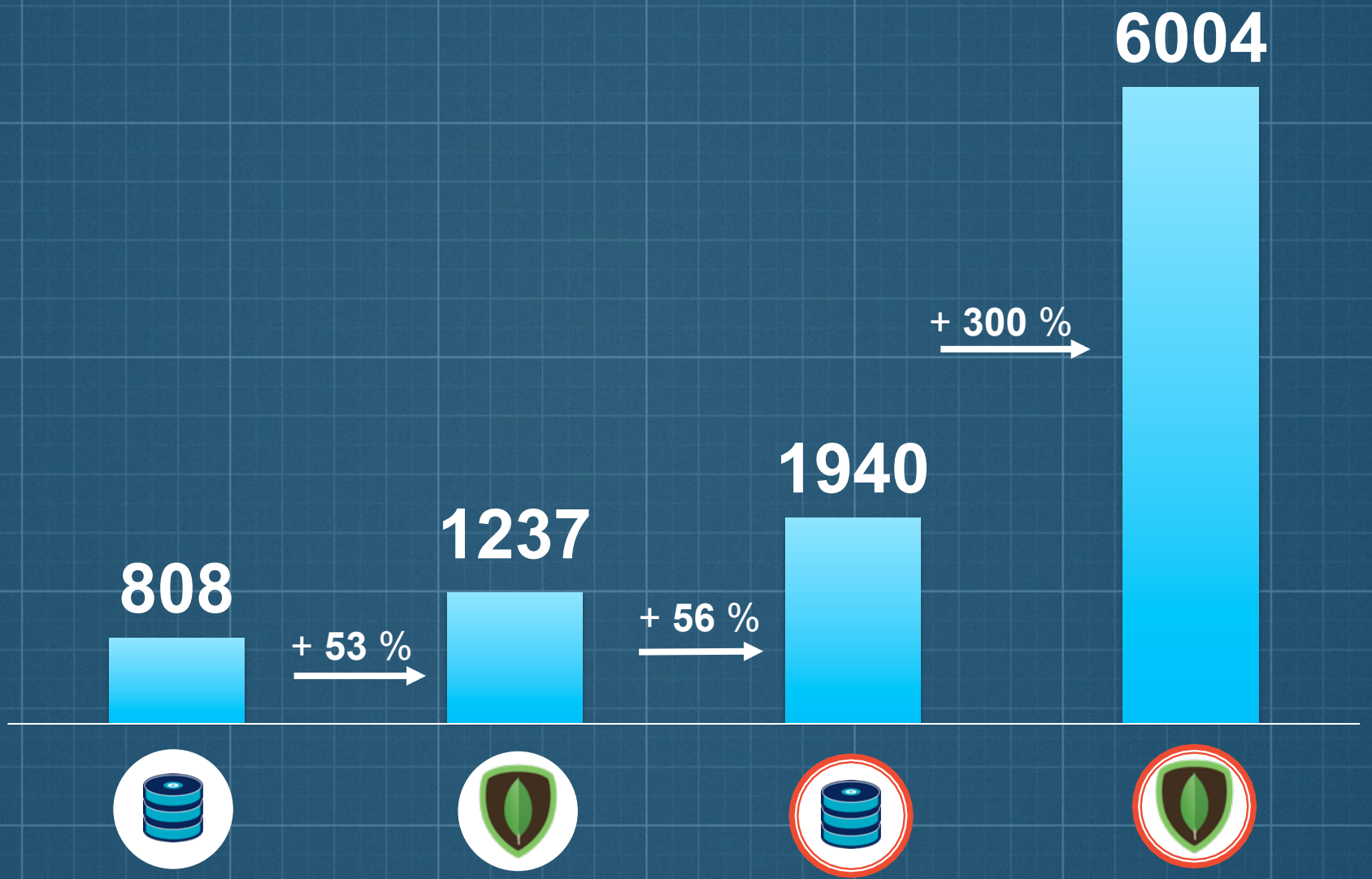
Performance
Redis

VS

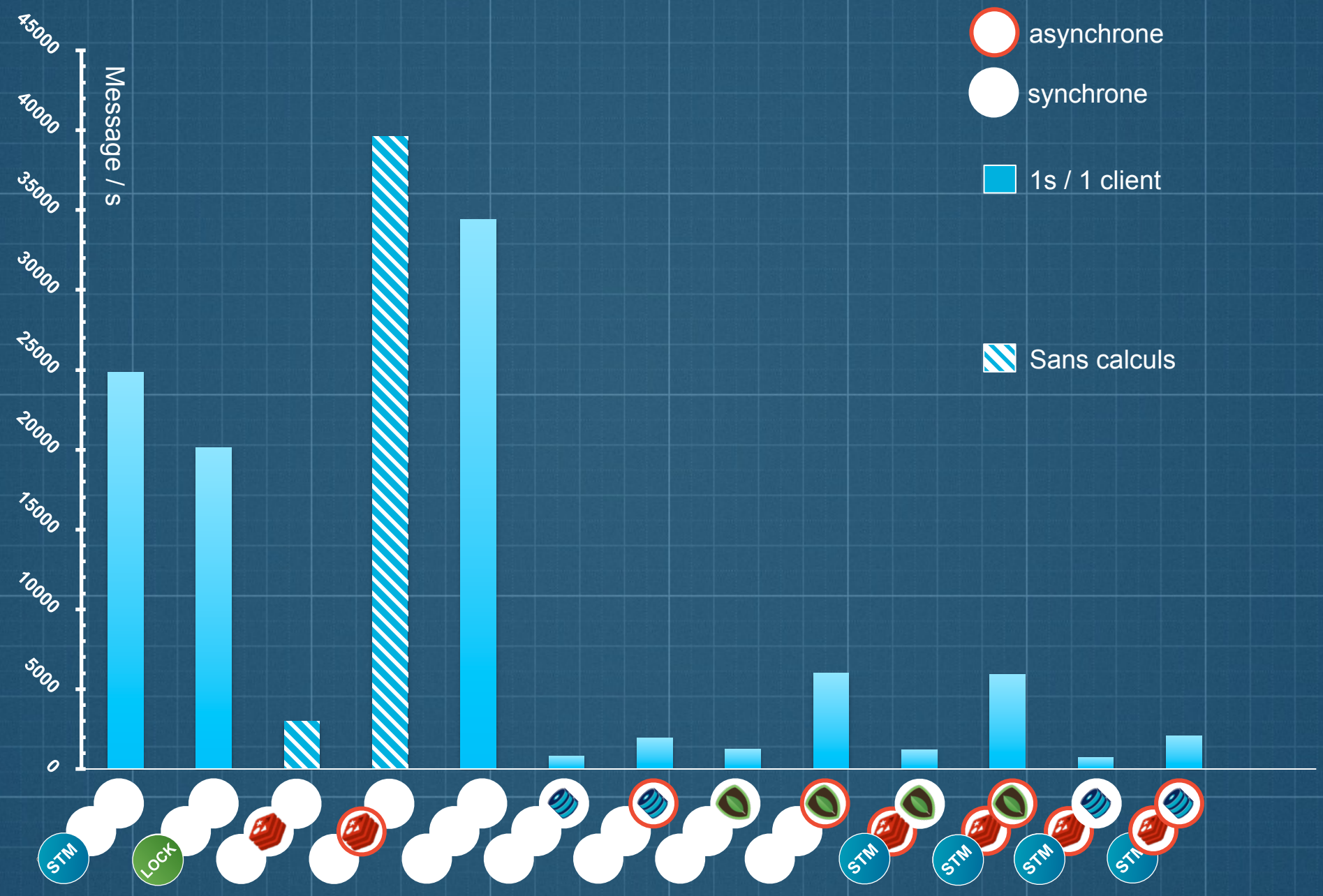
Performance
Redis



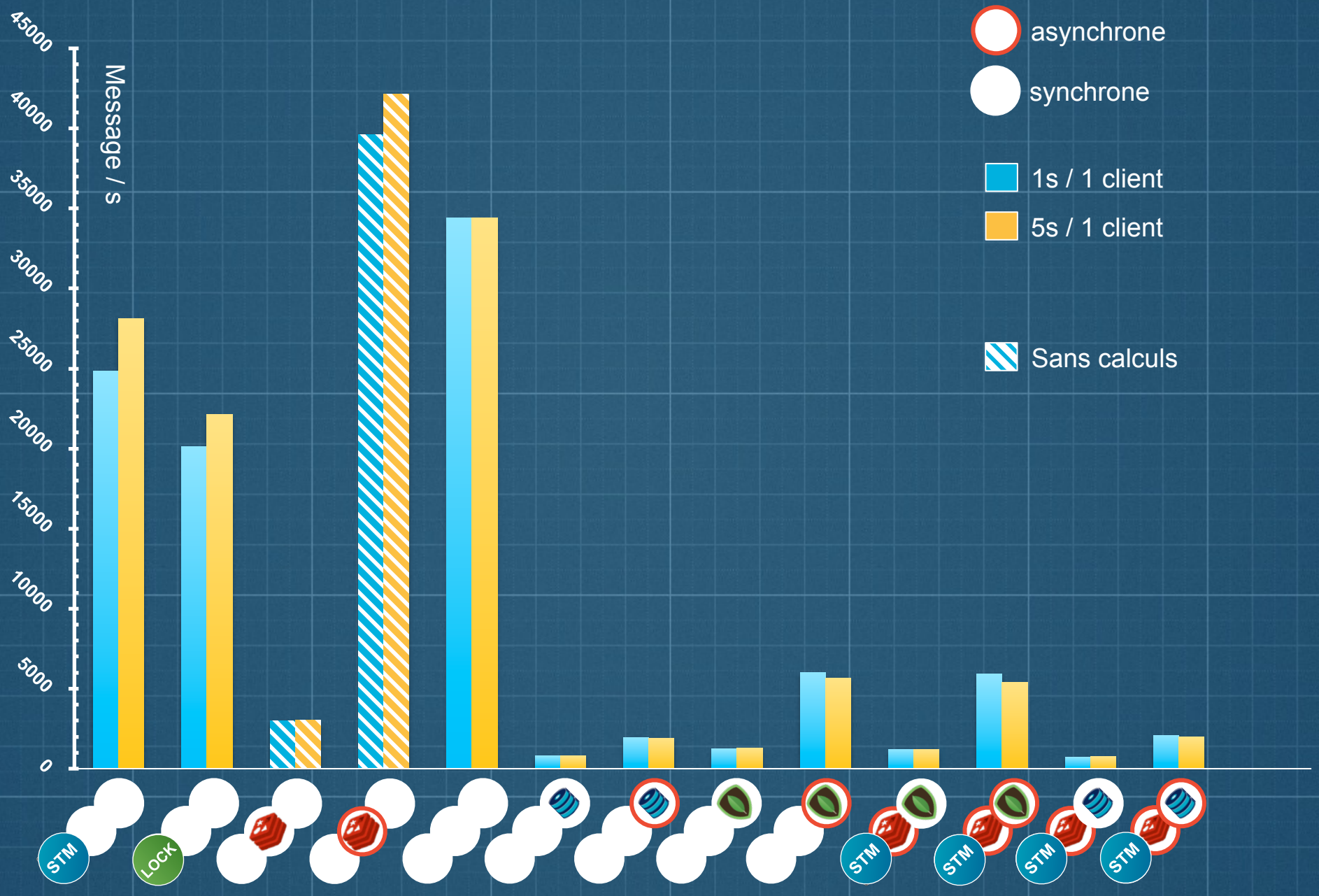
Performance
Mongo - MySQL



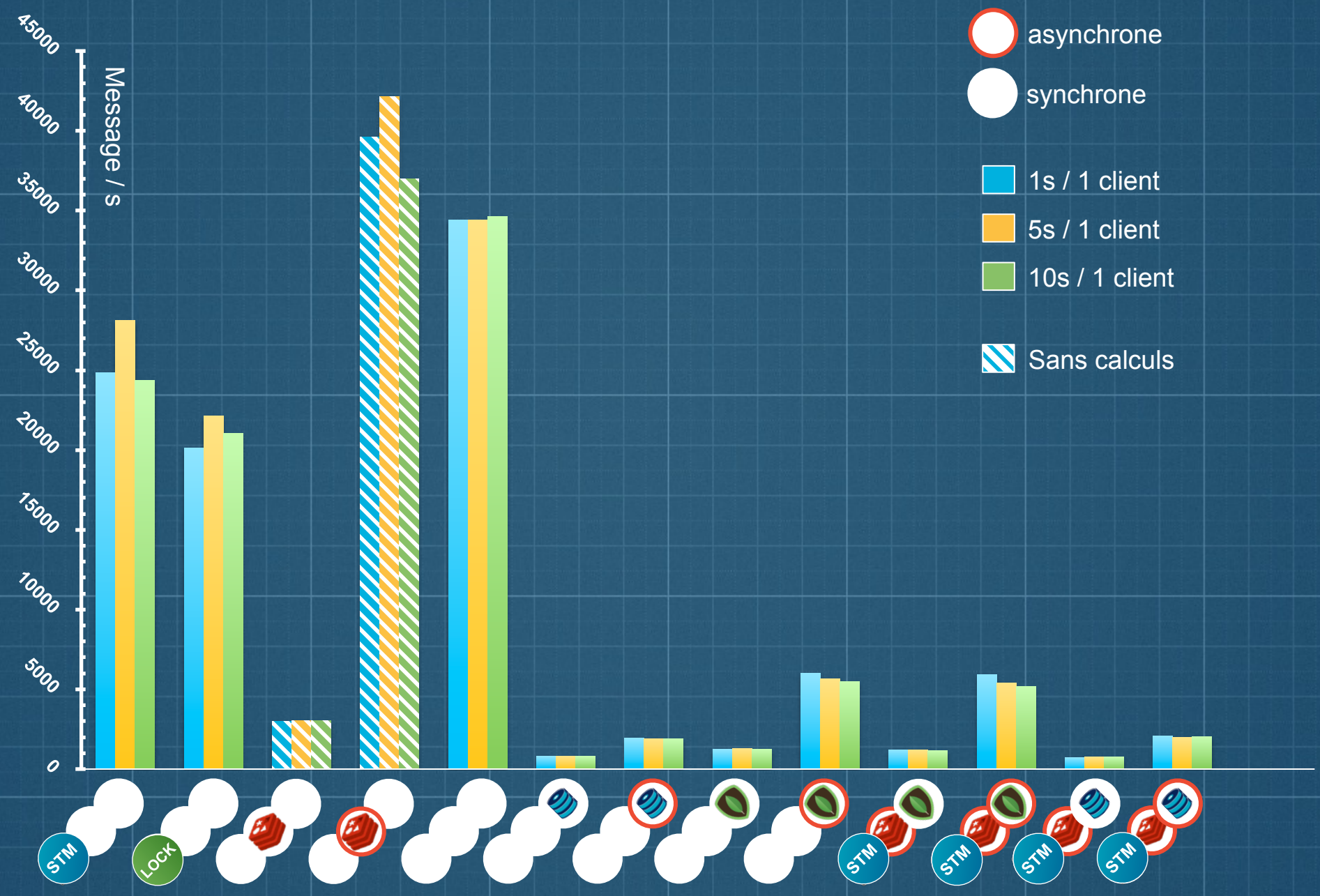
Injection auto-ajustés



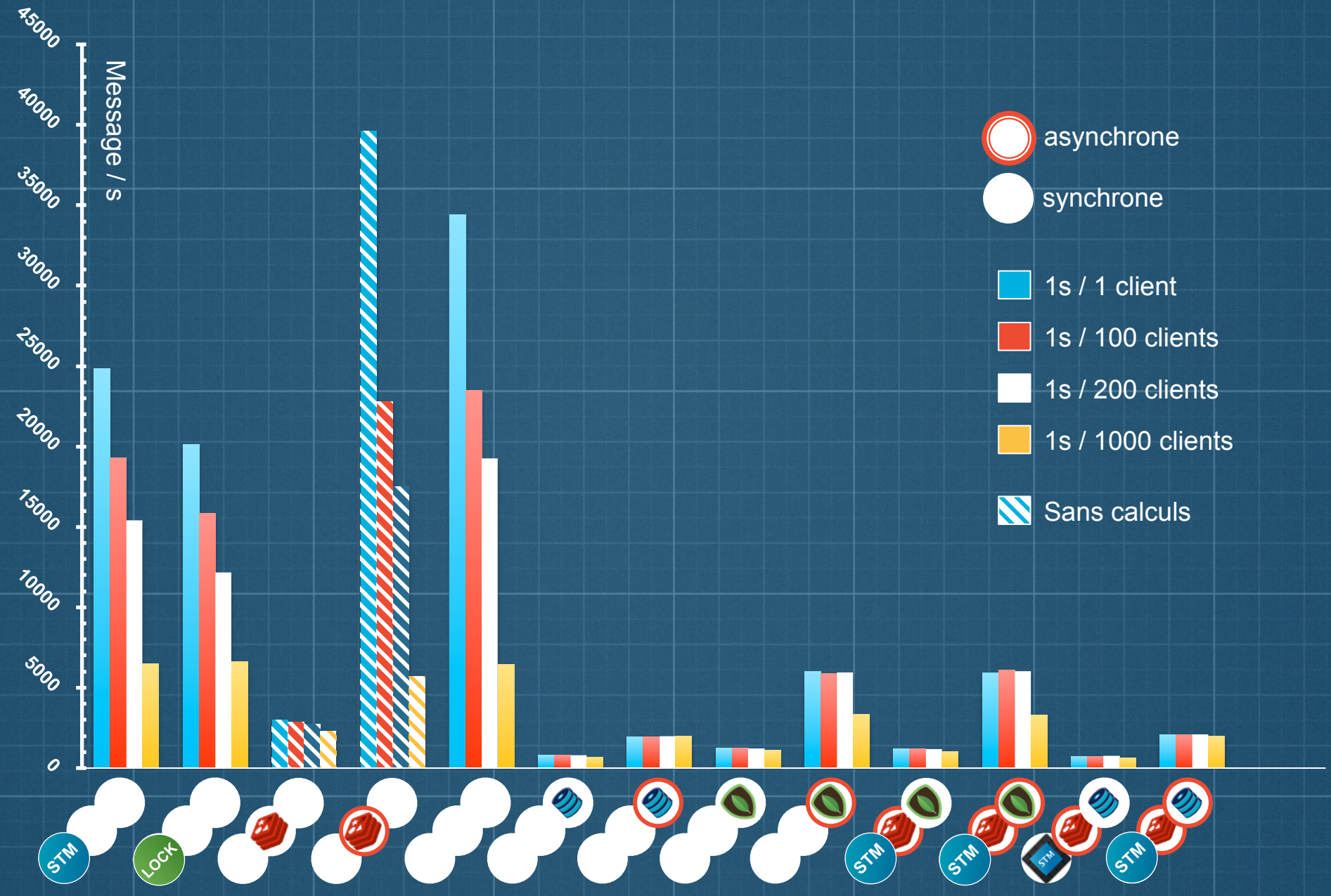
Injection auto-ajustés



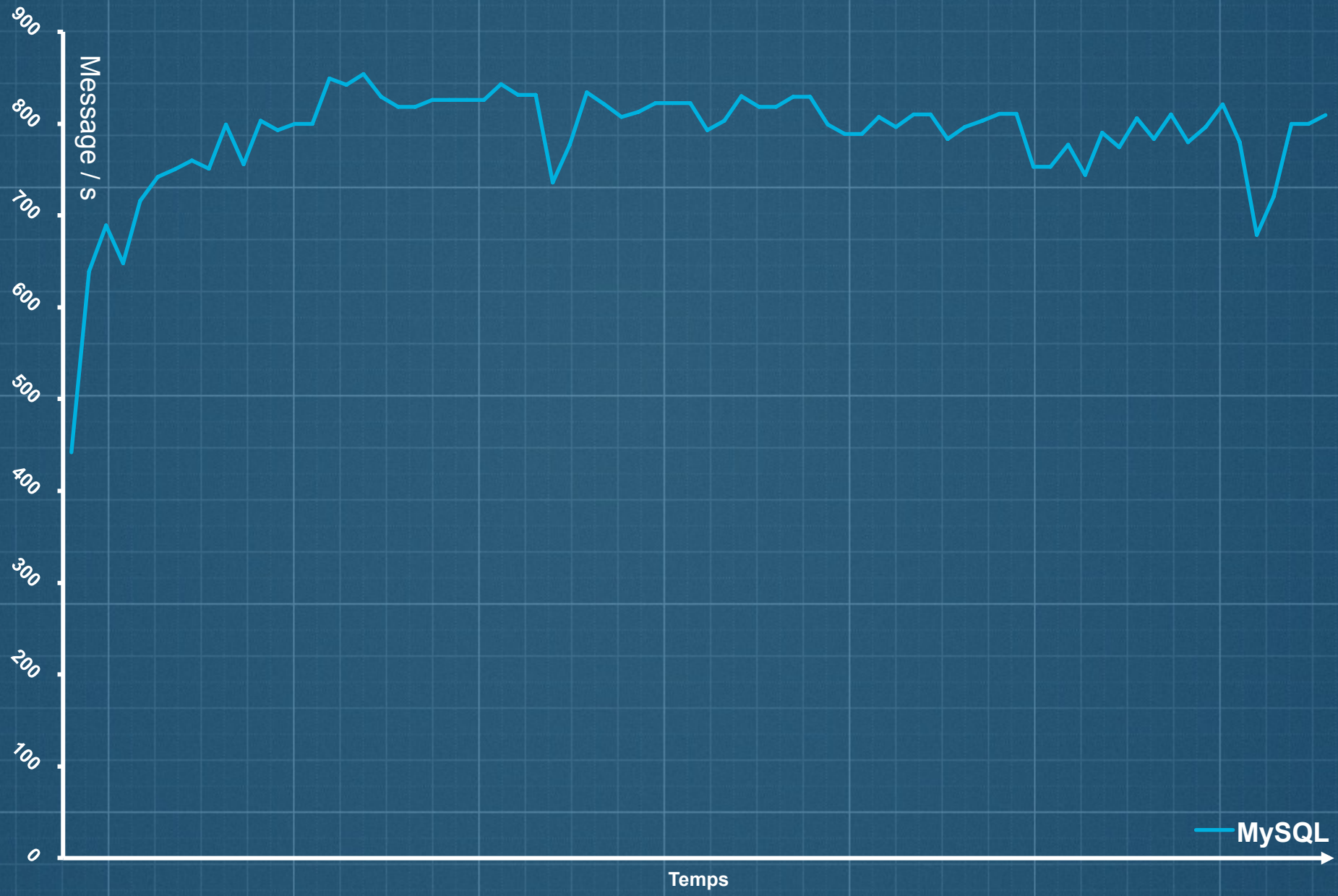
Injection auto-ajustés



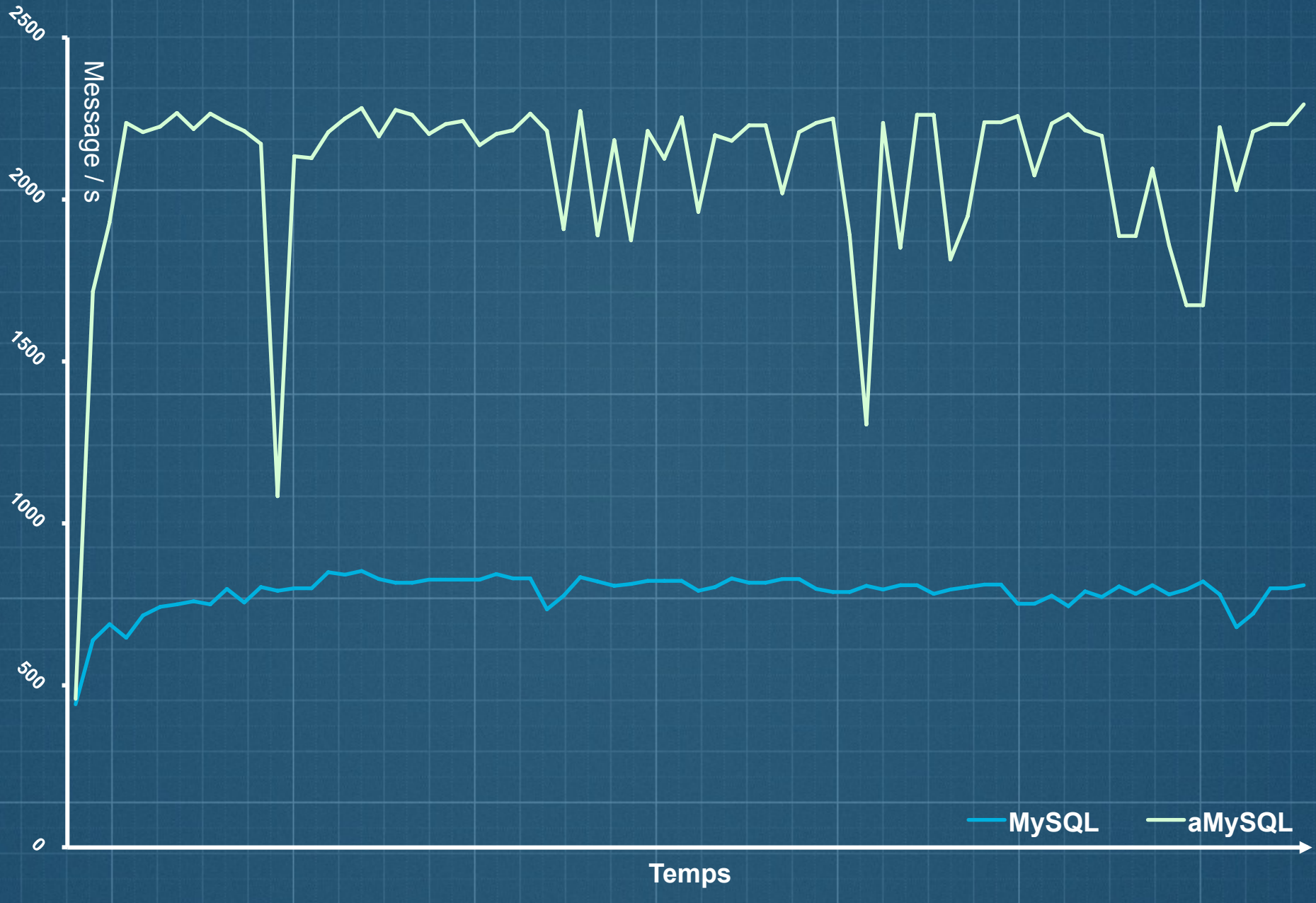
Injection auto-ajustés



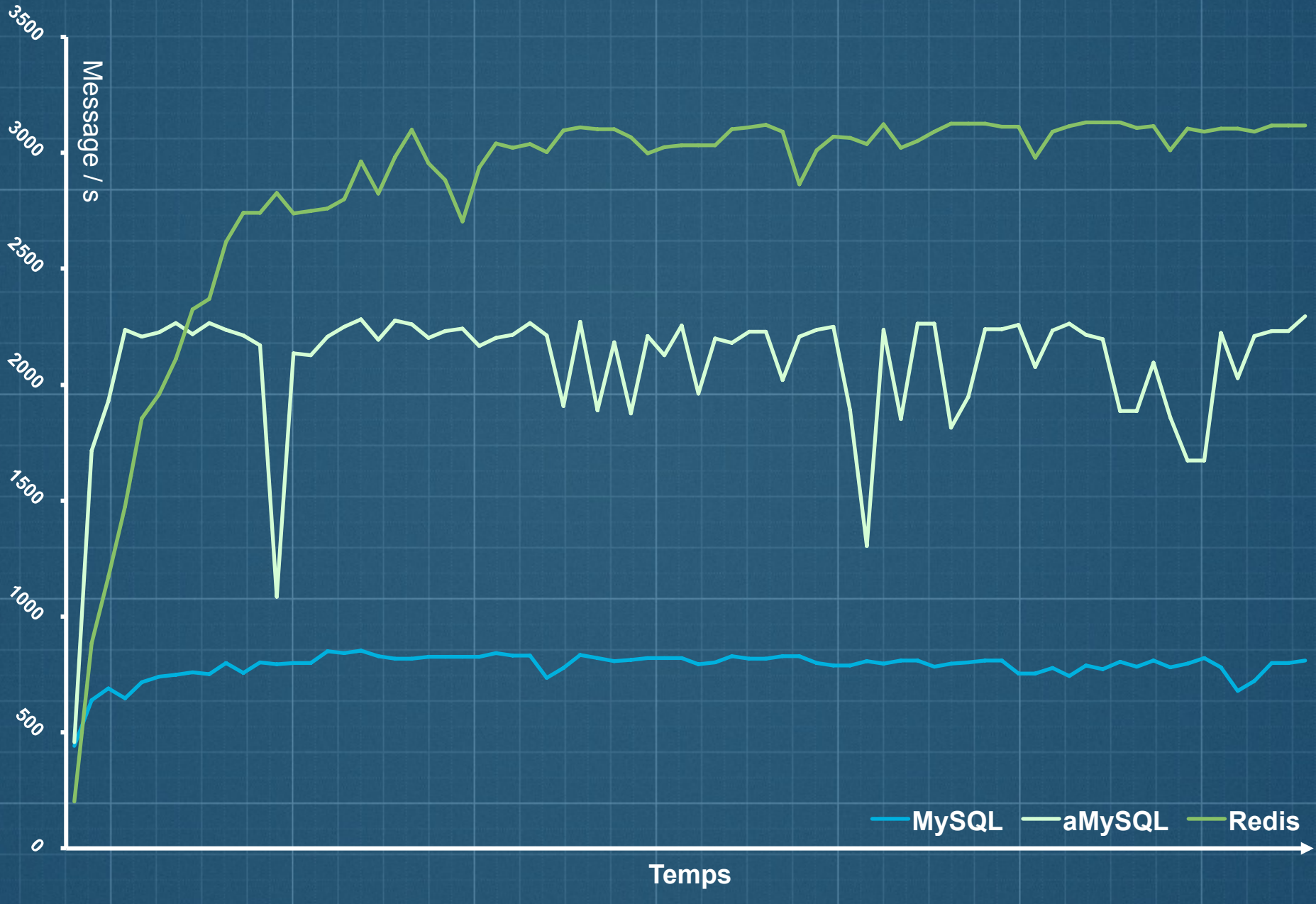
Evolution du débit



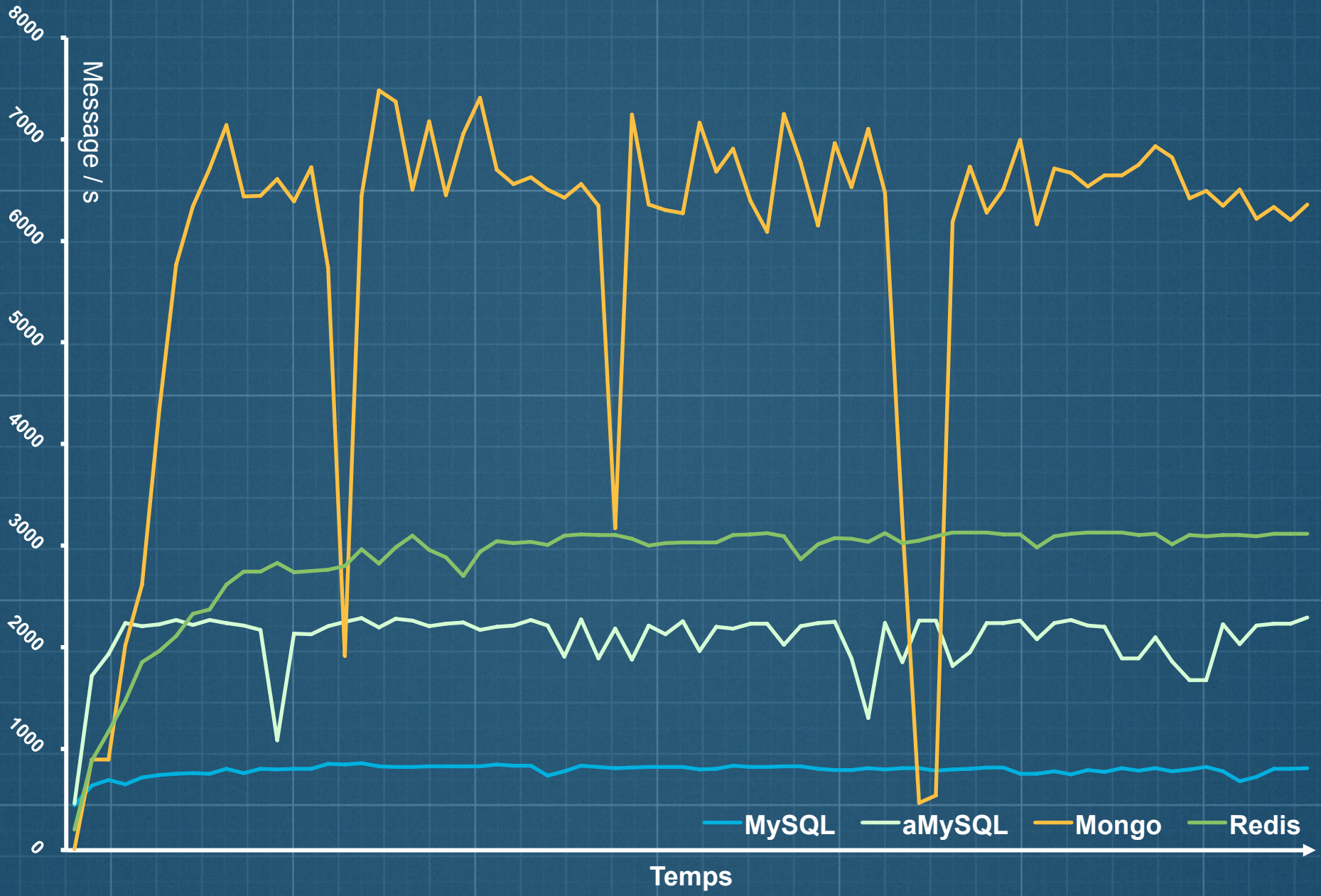
Evolution du débit



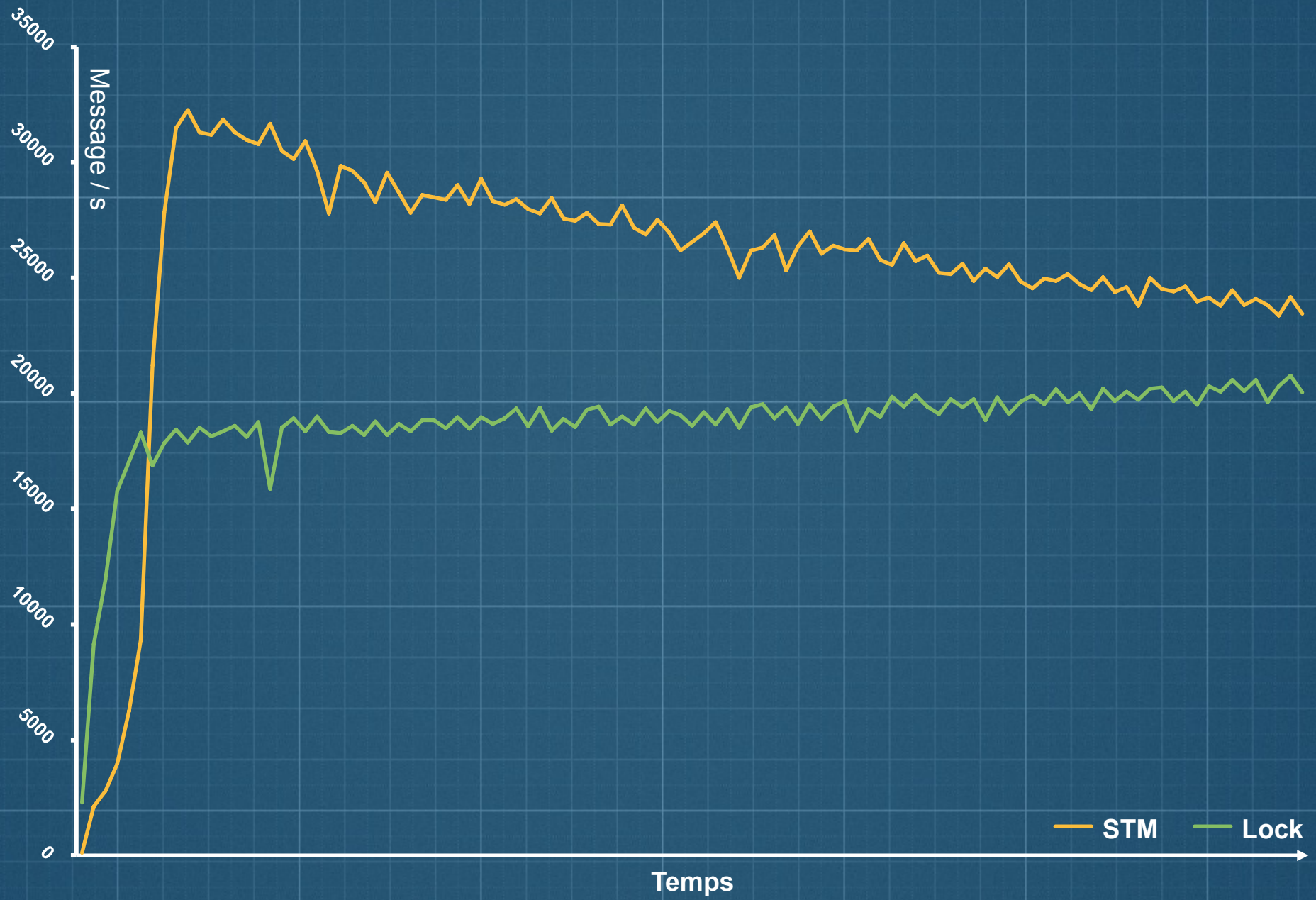
Evolution du débit



Evolution du débit



Evolution du débit



	80 %		90 %		100 %	
	Débit Effectif	Ecart	Débit effectif	Ecart	Débit effectif	Ecart
	83333	- 17 %	83333	- 26 %	83333	- 34 %
	27778	-	31250	-	32258	- 6 %
STM	20000	-	22222	-	24390	-
LOCK	16129	-	18182	-	20000	-
	2396	-	2726	-	3055	-
	31250	-	34483	-	37037	- 7 %
	435	- 33 %	439	- 40 %	436	- 46 %
	1550	-	1754	-	1873	-
	982	-	1086	-	1213	-
	4545	- 5 %	4237	- 22 %	4016	- 33 %
STM 	521	- 46 %	1065	-	1175	-
STM 	3984	- 16 %	4926	- 7 %	5208	- 12 %
STM 	421	- 29 %	426	- 36 %	422	- 43 %
STM 	1667	-	1832	-	1887	- 10 %

There is a better way

OCTO Technology

Comparatif Latence

Requêtes longues (10 000 records)

Lissage 200 ms entre client

Nb Clients	RAM			MONGO			AMYSQL			MYSQL		
	MIN	AVG	MAX	MIN	AVG	MAX	MIN	AVG	MAX	MIN	AVG	MAX
10	58	326	1312	145	817	1820	132	197	612	75	225	679
100	49	95	1316	13	199	1752	84	136	583	76	98	552
200	54	80	1307	67	167	1788	78	132	503	75	92	517
500	52	70	1291	38	147	1896	43	131	451	73	92	524

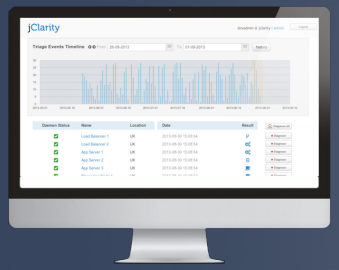
Garbage collector

- Dans ce type d'application l'ennemi premier est la latence
- Le principal responsable de la latence est le GC

Garbage collector

- Nous avons lancé l'injection sur une longue période.
- Full GC en nombre important
- Emballement

jClarity



APM - Application Performance Management

- Bottleneck detection entre autre

Problème détecté

- Too much context switching

Confirmation du problème

- Validé avec la commande **`vmstat`**

Solution

- Réduire le nombre de thread en mutualisant:

  Les pools de thread, les systèmes d'acteur, les pools Netty, le pool des futures

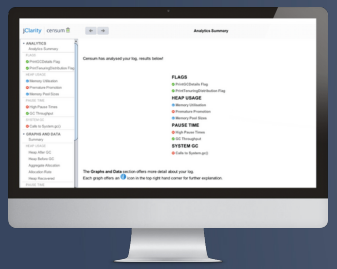
Optimisation GC 1/2

Première configuration

- `-Xmx4096M`
- `-XX:MaxNewSize=384m`
- `-XX:NewSize=384m`

Throughtput: 98.1 %

jClarity | Censum



- Analyse des logs GC
- GC problem detection
- Pas d'installation

Problème détecté

- Premature Promotion

  Les objets passent du young memory pool (eden, survivor) au tenured memory pool (old) de manière prématurée

Solution

- Augmenter l'eden et le survivor

Optimisation GC 2/2

Deuxième configuration

- `-Xmx4096M`
- `-XX:MaxNewSize=1500m`
- `-XX:NewSize=1500m`

Throughtput : 99.1 %

Deuxieme passe Censum: Aucun problème

Résultat Injection longue

RAM, Redis, Mongo

- 1 an de message dans l'ancien système
- 4 Go de données CSV
- 27 M de records
- Absorbé en 3h à 80 % de charge maximale

Conclusion

Conclusion

- Le réactif c'est efficace, même sur des bases traditionnelles
- Pas facile à tester (devrait s'améliorer)
- Combinaison de technologies -> pertinent
- Vous devrez passer par la case GC
- Reactive manifesto: www.reactivemanifesto.org

On aurait aimé

1. Paralléliser la distribution avec Akka
   Requêtes clients
2. Tester un cache distribué
   HazelCast
3. Comparer avec Cassandra
4. Utiliser Protobuf comme format de message

Questions

There is a better way

OCTO Technology