Table des matières

[1. Introduction 3](#_Toc59890695)

[1.1 Context 3](#_Toc59890696)

[1.1.1 Usual approach 3](#_Toc59890697)

[1.1.2 New approach 3](#_Toc59890698)

[1.2 CRDT 4](#_Toc59890699)

[1.2.1 Principles 4](#_Toc59890700)

[1.2.2 Advantages 4](#_Toc59890701)

[1.3 Lasp and Awset 4](#_Toc59890702)

[1.4 Goals 4](#_Toc59890703)

[1.4.1 Improved API 4](#_Toc59890704)

[1.4.2 Measurements 4](#_Toc59890705)

[1.5 Summary and structure 4](#_Toc59890706)

[2. Contributions 5](#_Toc59890707)

[2.1 Measurement tools 5](#_Toc59890708)

[2.1.1 Principle 5](#_Toc59890709)

[2.1.2 API 5](#_Toc59890710)

[2.1.3 Examples 5](#_Toc59890711)

[2.2 Adaptation tools 5](#_Toc59890712)

[2.2.1 Principle 5](#_Toc59890713)

[2.2.2 API 5](#_Toc59890714)

[2.2.3 Examples 5](#_Toc59890715)

[2.3 Scripts 5](#_Toc59890716)

[2.3.1 Static measurement scripts 5](#_Toc59890717)

[2.3.2 Dynamic measurement scripts 5](#_Toc59890718)

[2.3.3 Quality-of-life scripts 5](#_Toc59890719)

[2.4 Lasp minor additions 5](#_Toc59890720)

[2.4.1 Readme improvement 5](#_Toc59890721)

[2.4.2 Memory leak 6](#_Toc59890722)

[3. Measures 7](#_Toc59890723)

[3.1 Parameters 7](#_Toc59890724)

[3.2 Initial measures 7](#_Toc59890725)

[3.2.1 Number of Nodes 7](#_Toc59890726)

[3.2.2 Nodes distance 7](#_Toc59890727)

[3.2.3 CRDT size 7](#_Toc59890728)

[3.2.4 CRDT operation 7](#_Toc59890729)

[3.2.5 Partition 7](#_Toc59890730)

[3.2.6 Update speed 7](#_Toc59890731)

[3.3 Adapted Lasp 7](#_Toc59890732)

[4. Results and analysis 8](#_Toc59890733)

[4.1 Results 8](#_Toc59890734)

[4.1.1 Number of Nodes 8](#_Toc59890735)

[4.1.2 Nodes distance 8](#_Toc59890736)

[4.1.3 CRDT size 8](#_Toc59890737)

[4.1.4 CRDT operation 8](#_Toc59890738)

[4.1.5 Partition 8](#_Toc59890739)

[4.1.6 Update speed 8](#_Toc59890740)

[4.2 Analysis 8](#_Toc59890741)

[5. Conclusion 10](#_Toc59890742)

[5.1 Summary 10](#_Toc59890743)

[5.2 Analysis conclusion 10](#_Toc59890744)

[5.3 Personal opinion and methodology 10](#_Toc59890745)

[6. Bibliography 11](#_Toc59890746)

# 1. Introduction

This chapter presents the general content and context of this manuscript, describing what is Lasp, what are CRDTs and what are their innovative aspects. The goals of this master thesis and its main structure will also be briefly introduced.

## 1.1 Context

In today world, large-scale distributed applications are more and more common. These applications, to work correctly on multiple devices must share distributed variables, in other words, values that can be accessed and modified consistently from any node of the system. These variables may then be used by the application for thousands of different possible usages. A good example is the case of IoT small devices with captors and sensors collecting information such as temperature, light, pressure...

The way to handle these distributed variables is generally hidden to the end-user but can represent an important part of the application implementation requiring for the developer to consider consistency and distribution. This master thesis will focus on the way to handle these distributed variables considering a particularly innovative approach that was introduced around 2011 and of which the Ecole Polytechnique de Louvain research team has developed an experimental version called Lasp.

### 1.1.1 Usual approach

The most common way to handle distributed variables is to centralize them with a database. This means every node will connect to the database to access the variables. This is generally handled with an API for the developer to avoid overthinking on technical problems such as causality and consistency. These databases usually allow some interesting features such as atomic operations and log history but actually require some (hidden) heavy algorithms.

Furthermore, this kind of distributed structure usually relies on redundancy with replicated databases in multiple datacenters to achieve high scalability, adding more complexity to handle causality and operation order consistency between replicas, introducing Consensus algorithms.

Finally, since strong Consistency conflicts with Availability and Partition-tolerance (CAP theorem1), these systems have to choose between CP (strong Consistency and Partition tolerance but low Availability), AP (high Availability and Partition tolerance but weak Consistency) and CA (strong Consistency and Availability and no Partition tolerance). While a good part of the mainstream distributed applications goes for the AP model with a loss of Consistency, no ideal solution exists.

### 1.1.2 New approach

A totally different approach is to rely on peer-to-peer instead of the usual structure with databases. This means no database servers running heavy algorithms is required, instead the distributed variables are handled via messages exchanges between nodes. This new alternative relies on an innovative way to represent the distributed variables. As opposed to the usual approach where distributed variables are generally just values registered and updated in a specific database, variables will be represented as a specific data-structure called Conflict-free Replicated Data Types (CRDTs2). It is the key concept that will be detailed below to understand this new approach along with all its advantages.

## 1.2 CRDT

CRDT is for Conflict-free Replicated Data Type. The main idea is that it is an abstract data type

with an interface designed for replication on multiple nodes and satisfying the following properties3:

1. Any replica can be modified without requiring any coordination with any other replica.
2. Two replicas receiving the same set of updates reach the same deterministic state guaranteeing state convergence.

Even if this approach might look surprising at fist sight (since it does not involve recording the distributed variable state in a specific place such as a database), this new way to represent distributed variables introduced in 2011 is already used by some big compagnies such as Riot Games, TomTom, Bet365, SoundCloud and some others4.

### 1.2.1 Principles

Convergence is the key-concept to understand CRDT principle. To clarify this concept, let’s illustrate it with a very general example considering a single distributed variable:

1. Every node has a local state representing the distributed variable. This local state is a data-structure than contains values and metadata, it is called a CRDT. The specific structure is not relevant here since it depends on the type of CRDT. In other words, the specific structure is not the same if the nodes share a variable representing a counter, a set of elements, a Boolean…
2. A node can adapt its local state to modify the variable without requiring any coordination with other nodes. For example, if the variable represents a set, it can add an element in it. When doing such, it will modify the values in the data-structure (CRDT) as well as the metadata.
3. From time to time, the nodes will send their local state to their peers. In other words, they will send their own version of the CRDT to their peers. When receiving such a message, the node will merge the received state with its own local state. The way this merge is implemented is very important since it is this specific operation that will guarantee the system convergence. Indeed, the merge uses the metadata to determine how to merge the two versions in a deterministic way representing the most causally recent modifications. This allows the most recent modifications to propagate from peer-to-peer to the entire system and eventually reach a consistent state on every node.

As mentioned above, the key-concept here is the convergence. It is the fact that, automatically, due to the CRDT metadata and the merge implementation, all the nodes will eventually reach the same consistent state.

### 1.2.2 Advantages

The incredible part is that the convergence described above is automatic, deterministic, independent of the received messages order (scheduler) and does not require any consensus algorithm other than simple metadata comparison. In other words, based on messages received from its peers, the node will determine how to update its local state, efficiently handling the distributed variable without requiring heavy algorithms or database. Cherry on the cake, it also makes it automatic to handle partitions.

* **Automatic**: The synchronization is pretty simple and straight forwards since the nodes send their local state regularly and automatically update their states based on peers messages.
* **Deterministic**: A set of received messages will always update the local state in the same way, resulting in the same final state.
* **Independent of the message order**: The merge operation will compare the received metadata with the local metadata to determine how to update the local state. When receiving, for example, a recent message followed by an old message, the node will update its local state based on the recent message and will just ignore the older message since its metadata are older than its own updated metadata. In other words, the message order has no impact since the merge operation will follow causality handled by metadata and not the receiving message order. Furthermore, since the implementation is state-based (the messages represents a state, not an operation), potentially lost messages are not a problem either since the most recent message represents the most recent state and does not require previous messages to be correctly interpreted.
* **No consensus required**: Simple metadata comparison within the merge operation allows the receiver node to easily determine how to update its state. No database server is required, consensus algorithm either.
* **Partition-tolerant:** The previous properties, especially the fact that message order and lost messages do not impact converge, allow to easily handle partition-tolerance. Indeed, when a node is temporarily unreachable, it will continue to work with its own state which might be temporarily inconsistent with other nodes. Then, when the partition is resolved, it will receive state messages from other nodes and directly update its local state to represent the most recent version.

Les avantages  
Expliquer c’est probablement pour cela que ça se démocratise et que plusieurs grandes compagnies l’ont adopté.

## 1.3 Lasp and Awset

Expliquer ce qu’est Lasp. Ce qu’est le Awset (en expliquant le orswot) et pourquoi j’ai décidé d’utiliser ça.

## 1.4 Goals

### 1.4.1 Improved API

Developper une API pour visualiser le convergence time.  
Developper une API pour modifier ce convergence time.

### 1.4.2 Measurements

Lasp est-il vraiment utilisable en pratique ?  
Donne-t-il de bons résultats ?  
Quels sont les éléments qui pourraient impacter son convergence time ?  
Quels sont, parmis ces éléments, ceux que j’ai décidé d’analyser ? Pourquoi ?

## 1.5 Summary and structure

Résumé général ce qui vient d’être dit en introduction.  
Dévoilement de la structure générale du manuscrit avec petite mise en bouche (micro dévoilement de la conclusion).

## 2. Contributions

Introduire le fait que cette section développe ce que j’ai codé et apporté à Lasp.

## 2.1 Measurement tools

### 2.1.1 Principle

### 2.1.2 API

### 2.1.3 Examples

## 2.2 Adaptation tools

### 2.2.1 Principle

### 2.2.2 API

### 2.2.3 Examples

## 2.3 Scripts

Expliquer que cette section va résumer les différents scripts.

### 2.3.1 Static measurement scripts

Expliquer ce que ça fait. Le fait que ce sont les premiers scripts que j’ai utilisé pour mes premières mesures, qu’ils sont assez basiques mais qu’ils permettent déjà de donner une rapide idée des performances.

### 2.3.2 Dynamic measurement scripts

Expliquer qu’il s’agit d’une version déjà un peu plus évoluée qui fait des mesures en continue.

### 2.3.3 Quality-of-life scripts

Expliquer qu’il y a tout une série de scripts qui permettent simplement de faciliter et accélérer certaines manipulations comme : recompiler Lasp en entier, créer un cluster de 5 nodes, mesurer l’utilisation mémoire des process, etc…

## 2.4 Lasp minor additions

### 2.4.1 Readme improvement

Expliquer que j’ai rencontré parfois quelques petits problèmes, asez simples à résoudre mais qui, de mon point de vue, pourraient être explicités sur le github officiel. Example : Préciser qu’il peut être nécessaire de partager au préalable un fichier erlang.cookie. Ouvrir une pullrequest proposant d’améliorer le readme.

### 2.4.2 Memory leak

Expliquer le souci de mémoire avec la table ets et le fait que ça stockait une redondance inutile qui finissait par faire exploser le process. Expliquer l’impact que cela avait et la façon dont ça a été résolu.

# 3. Measures

Expliquer ce qui est mesuré (convergence time, memory usage, nombre de messages/sec).  
Rappeler que j’ai décidé d’utiliser le awset (orswot) pour toutes mes mesures. Faire un petit rappel sur ce qu’est Lasp, ce qu’est le Awset et ce qu’il permet.   
Expliquer le mode opératoire (quels scripts ont été utilisés), combien d’itérations ont été réalisées, etc.  
Expliquer que cette partie a pour but d’expliquer ce qui a été mesuré et comment. Il s’agit simplement de paragraphes expliquant comment j’ai mesuré telle ou telle chose. Les résultats seront dévoilés au chapitre suivant (graphes).

## 3.1 Parameters

Expliquer les paramètres sur lesquels j’ai joué (nombre de nodes, nombre d’éléments dans le CRDT, distance entre les nodes, add ou remove, update speed, partition).

## 3.2 Initial measures

Expliquer qu’il s’agit là des mesures réalisées sur Lasp tel qu’il était fourni sur le github officiel avant d’avoir développé l’API pour modifier le convergence time.

### 3.2.1 Number of Nodes

### 3.2.2 Nodes distance

### 3.2.3 CRDT size

### 3.2.4 CRDT operation

### 3.2.5 Partition

### 3.2.6 Update speed

## 3.3 Adapted Lasp

Expliquer qu’il s’agit maintenant des mesures qui ont été faites après avoir au préalable modifié le convergence time via l’API que j’ai développé. Par exemple, on a pris par défaut un interval de 100ms plutôt que l’initial 10000ms.  
Expliquer que toutes les mesures décrites plus haut ont également été réalisées avec un interval plus court.

# 4. Results and analysis

Expliquer que dans cette partie tous les résultats vont être dévoilés.  
Toutes les mesures ont été réalisées sur base du Lasp initial (tel que fournis sur le github officiel) puis également sur une version pour lequel le state\_interval a été réduit à 100ms.  
En suite une analyse sera faite.

## 4.1 Results

### 4.1.1 Number of Nodes

Dévoiler le graphe pour initial\_lasp 5,10,15,20 nodes.  
Dévoiler le graphe pour final\_lasp 5,10,15,20 nodes (c’est-à-dire la version avec 100ms de interval).  
Discuter à propos des graphes. Essayer de comprendre ce qui cause cela.

### 4.1.2 Nodes distance

Meme principe que précédemment

### 4.1.3 CRDT size

Meme principe que précédemment

### 4.1.4 CRDT operation

Meme principe que précédemment

### 4.1.5 Partition

Même principe que précédemment.

### 4.1.6 Update speed

Même principe que précédemment.

## 4.2 Analysis

Résumer ce qu’on a appris sur base de tous les graphes.  
Revenir sur le fait que globalement, l’API pour modifier le convergence time fonctionne.  
Essayer d’expliquer ce qui pourrait (faire des hypothèses) expliquer telle ou telle comportement (pourquoi c’est plus lent quand il y a plus d’éléments par exemple ?).

# 5. Conclusion

## 5.1 Summary

Rappeler donc ce qu’est initiallement Lasp. Ce que j’ai apporté à Lasp. Ce que j’ai mesuré et découvert.

## 5.2 Analysis conclusion

Expliquer ce que les résultats semblent dire de Lasp. Plot un graphe montrant la limite de ce que Lasp peut faire (udpate speed en verticale, convergence time en horizontal).  
Expliquer qu’avec l’API il est possible de configurer Lasp selon les besoins si on veut une convergence plus rapide (au risque de surcharger un peu le réseau) ou pas etc…  
Expliquer les différentes sections sur ce graphe.

## 5.3 Personal opinion and methodology

Expliquer ce que je pense personnellement de Lasp. Est-ce que je l’utiliserais personnellement. Expliquer que je trouve cela encore assez expérimental (pour le cas de Lasp) bien que prometteur car certaines grandes entreprises utilisent déjà du CRDT, etc…  
Expliquer comment j’ai travaillé, chaque semaine, avec mes réunions hebdomadaires avec Peter Van Roy etc.

# 6. Bibliography

1. CAP theorem (trouver une bonne source)

2. CRDT

3. document : crdts\_overview

4. source montrant que Rio Games, TomTom, etc utilisent des CRDTs.