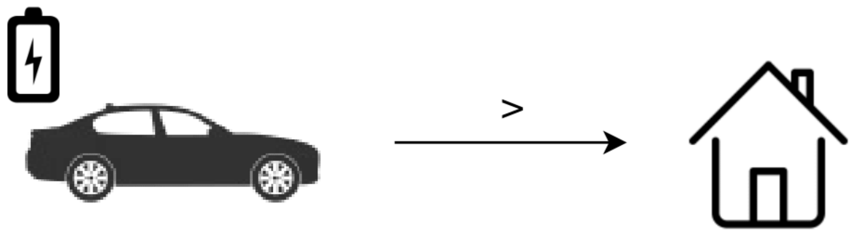
Distributed Convex Optimization Framework based on Bulk Synchronous Parallel (BSP) model

Presenter: Behroz Sikander
Supervisor: Prof. Dr. Hans-Arno Jacobsen
Advisor: Dipl.-Ing. Jose Adan Rivera Acevedo

Date: 4th March, 2016

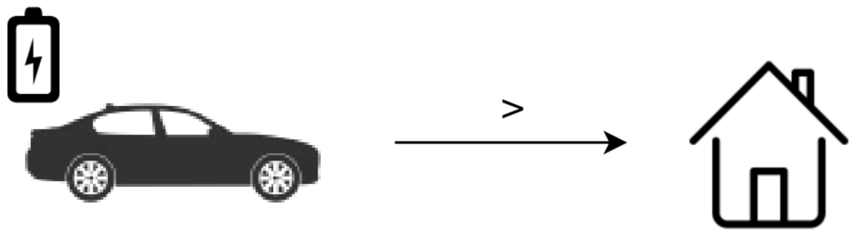
Trend towards electric vehicles

Trend towards electric vehicles

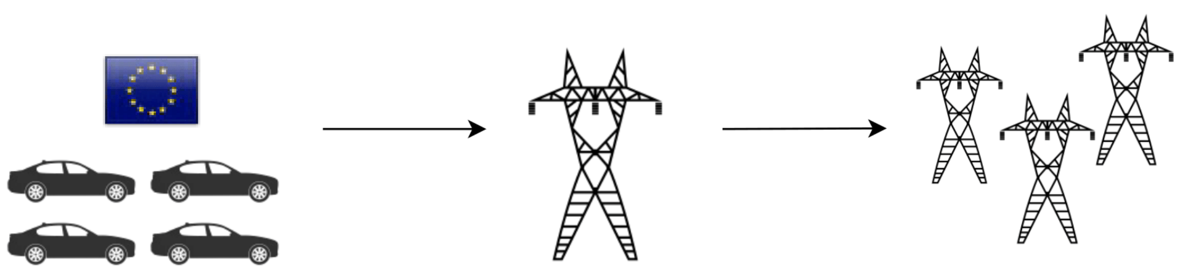


Electric vehicles takes more power than a house

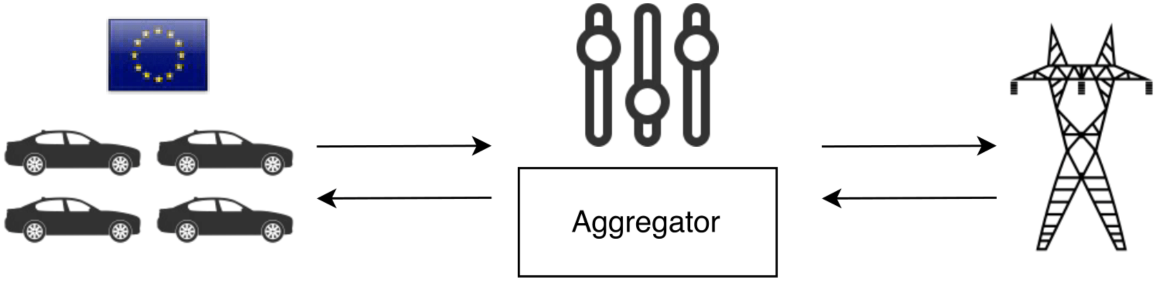
Trend towards electric vehicles



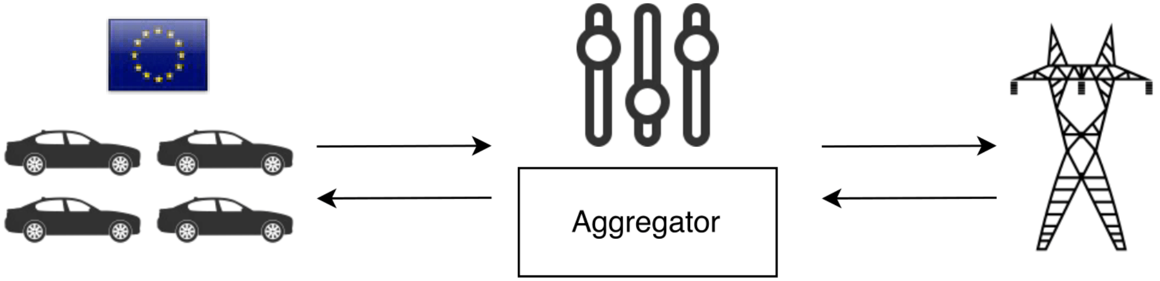
Electric vehicles takes more power than a house



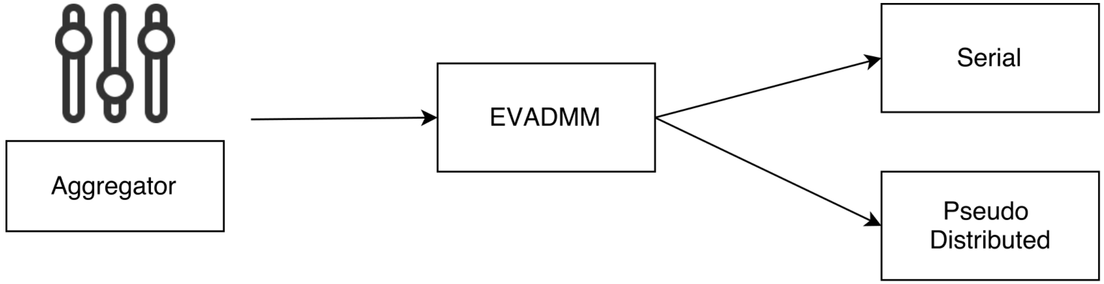
9 Mio EVs -> Grid overloaded! Increase Infrastructure.



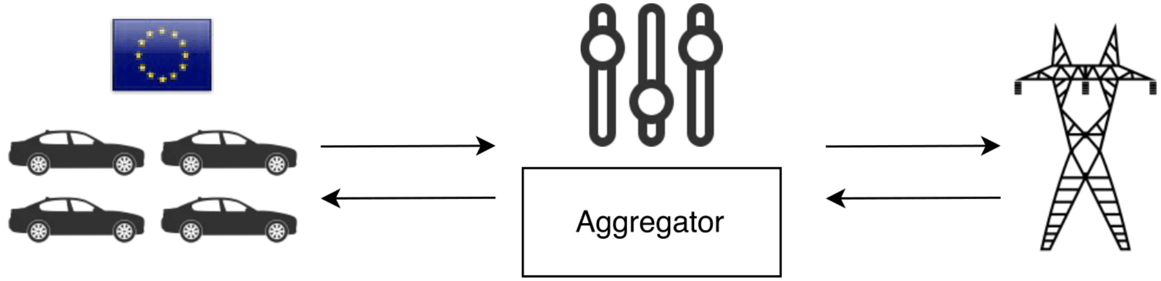
Keep infrastructure. Controlled EV charging



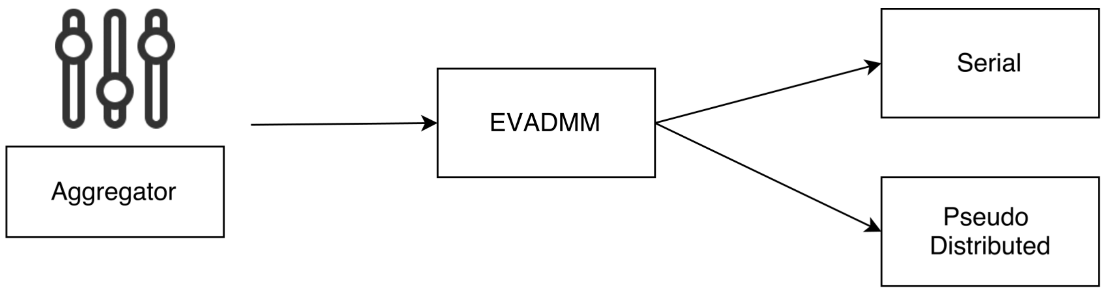
Keep infrastructure. Controlled EV charging



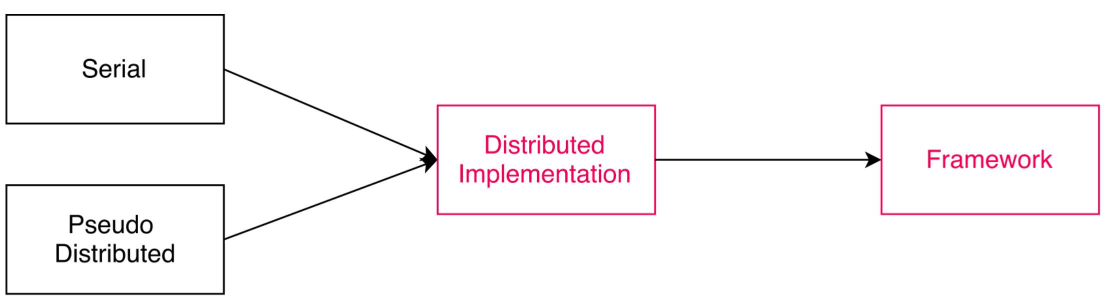
Proposed solution -> EVADMM



Keep infrastructure. Controlled EV charging



Proposed solution -> EVADMM



Thesis Goal: Distributed implementation & Framework

“

Total time to process N EVs using M machines ?

Total machines required to process N EVs in T time ?

Agenda

- Background
- Algorithm
- Deployment
- Results
- Framework

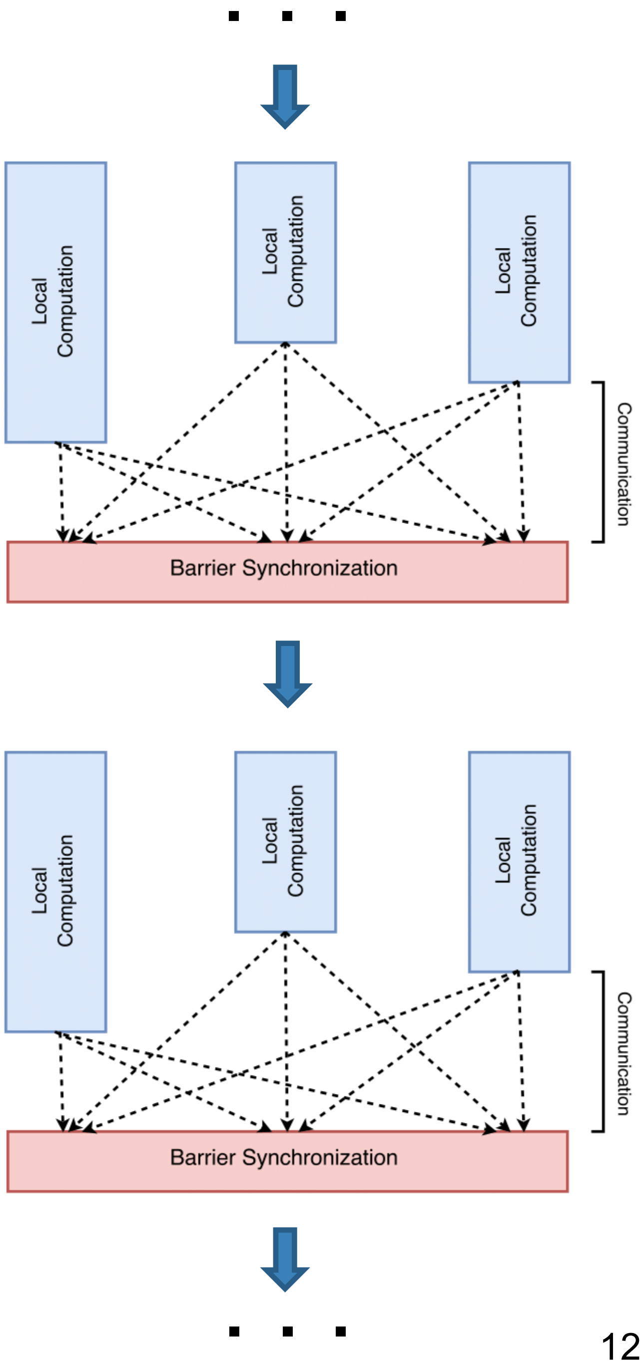
Background

BSP

- Bulk Synchronous Parallel
- Developed by Leslie Valiant in 1980s
- *Model for designing parallel algorithm*
- Strong theoretical background
- BSP computer has
  - p processors
  - each with local memory
  - point-point communication

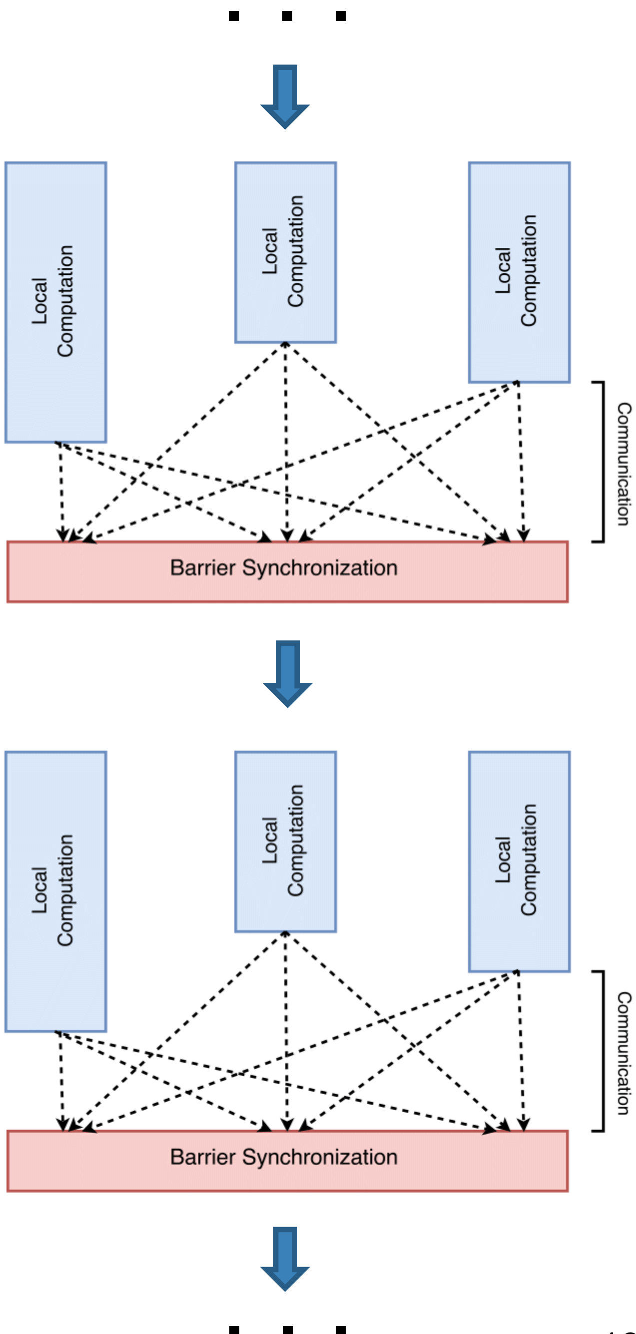
Supersteps

- Computation divided in *supersteps*
  - concurrent local computation (w)
  - global communication (h)
  - barrier synchronization (l)



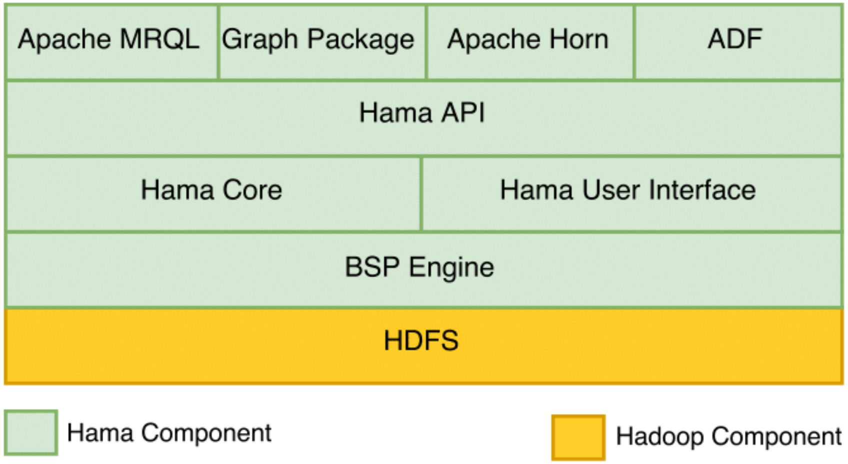
Supersteps

- Computation divided in *supersteps*
  - concurrent local computation (w)
  - global communication (h)
  - barrier synchronization (l)
- Total runtime of all supersteps
  - W + H.g + S.l
  - where W is the total local comp. time
  - g = time to deliver a message
  - H.g = total time for communication
  - S = total supersteps
  - S.l = Total time required for barrier synchronization



Apache Hama

- Opensource, in-memory and iterative
- Distributed computation framework
- BSP programming model
- Processes data stored in HDFS
- Replacement of MapReduce
- Can process massive datasets
- Clean programming interface



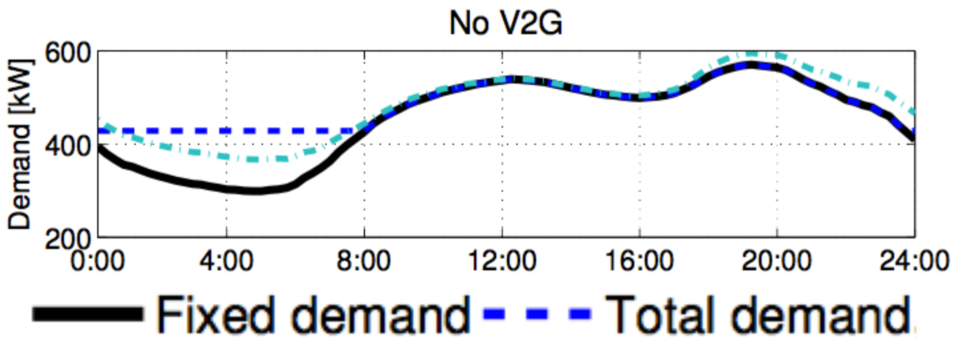
HDFS

- *Distributed storage*
- Runs on *commodity hardware*
- Master-Slave model (Namenode and DataNode)
- Files are broken down into blocks of 128MB.
- Can store huge amounts of data
- Scalable, available and fault tolerant
- This thesis : Used to store data

Algorithm

Controlled charging has to strike a balance between

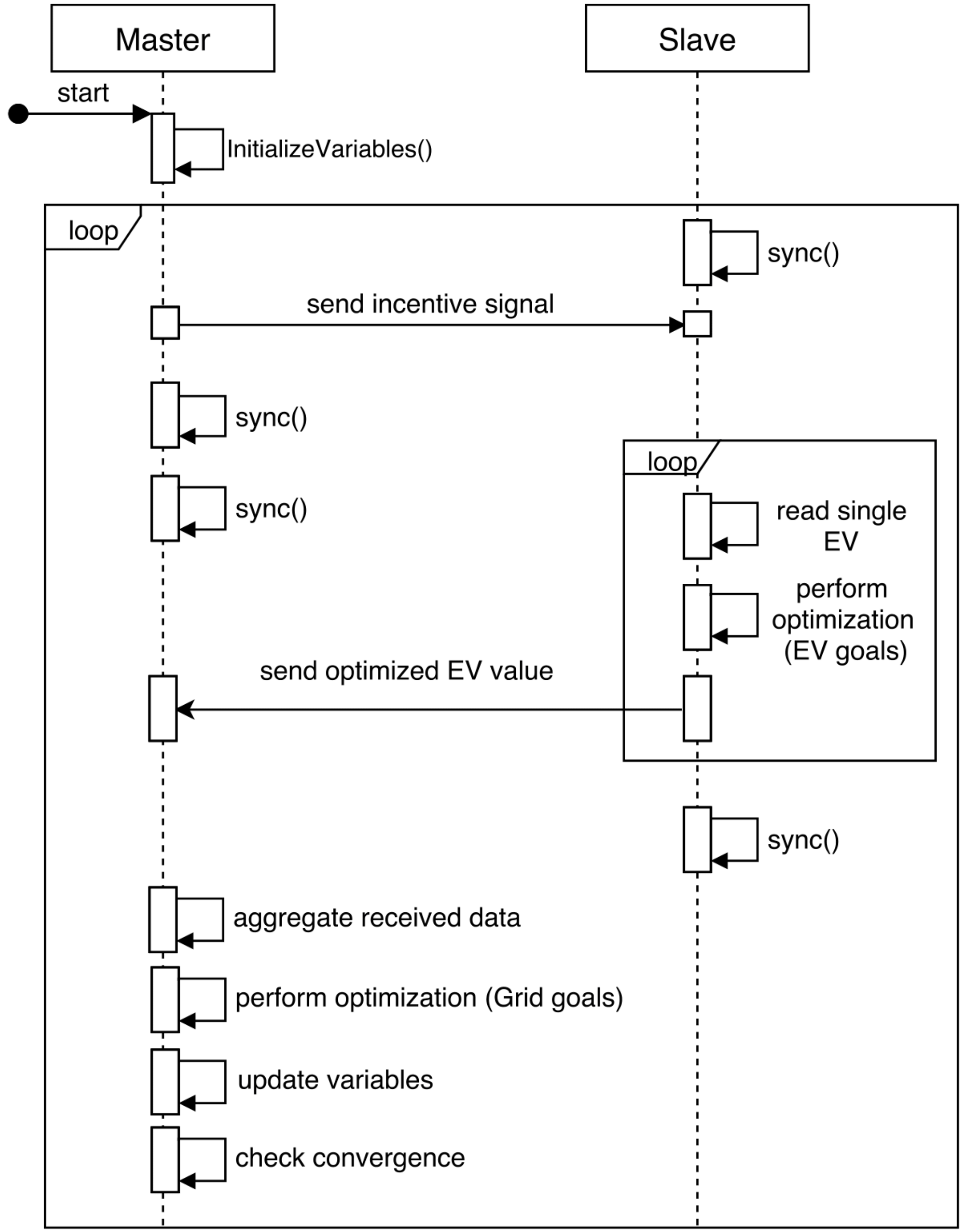
- Grid goals (Valley Filling)
  - EV charging in valleys of fixed demand
  - Avoid peaks
  - Leads to stable system



- EV goals (EV Optimization problem)
  - Minimize battery depreciation

Algorithm

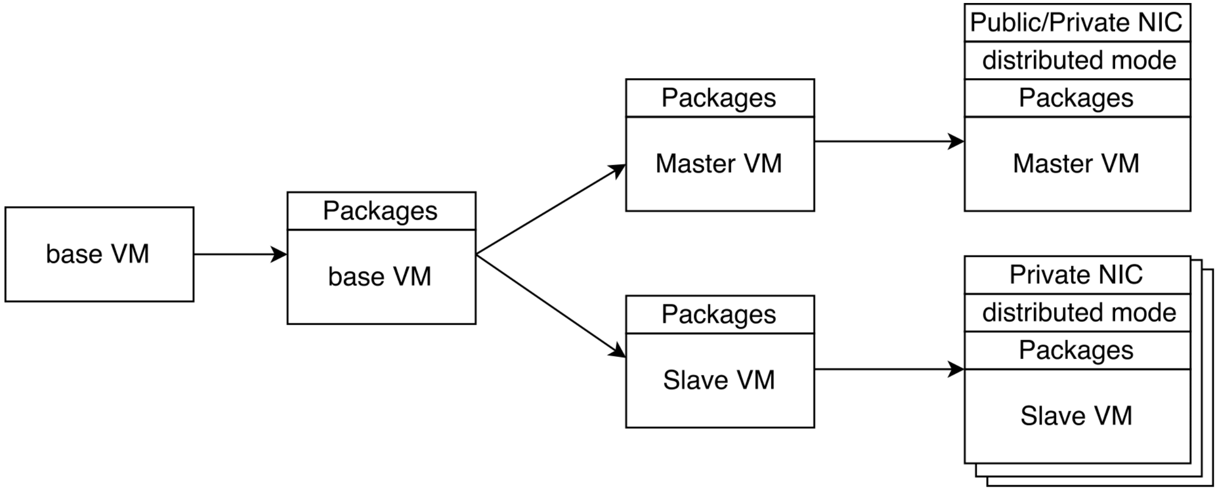
- Master executes Grid goals
- Slave executes EV goals
- sync used for barrier synchronization
- 1 Master, N Slave tasks



Deployment

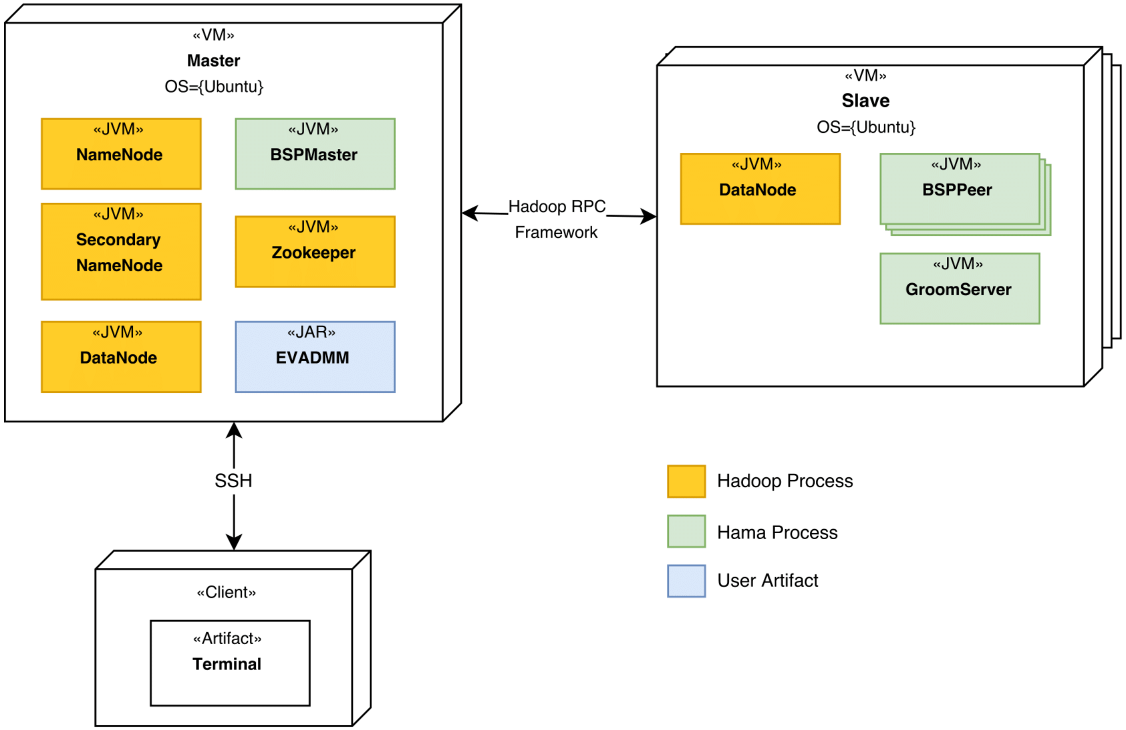
LRZ

- Leibnitz RechenZentrum
- Infrastructure as a Service
- Flexible, secure, highly available
- Used 10 VMs (Ubuntu, 4 CPU, 8GB RAM)
- Installed Hadoop, Hama and CPLEX in distributed mode



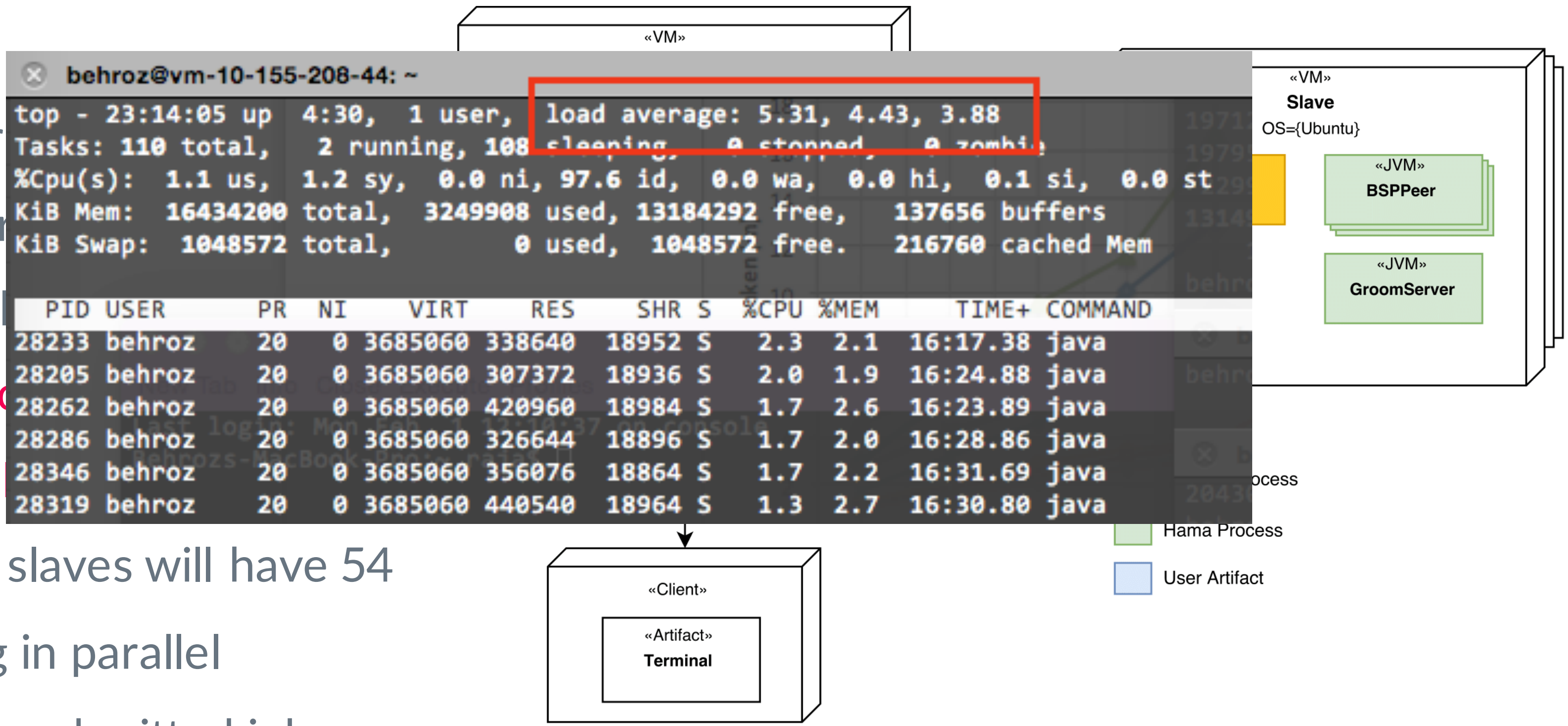
Deployment

- Client submits the job to BSPMaster
- BSPMaster communicates with GroomServer to start multiple BSPPeer (tasks).
- Each slave can run upto 5-6 tasks in parallel.
- Example: 9 slaves will have 54 tasks running in parallel executing the submitted job.



Deployment

- Client submits the job to BSPMaster
- BSPMaster
  GroomServer
  BSPPeer (tas
- Each slave c
  tasks in paral
- Example: 9 slaves will have 54 tasks running in parallel executing the submitted job.

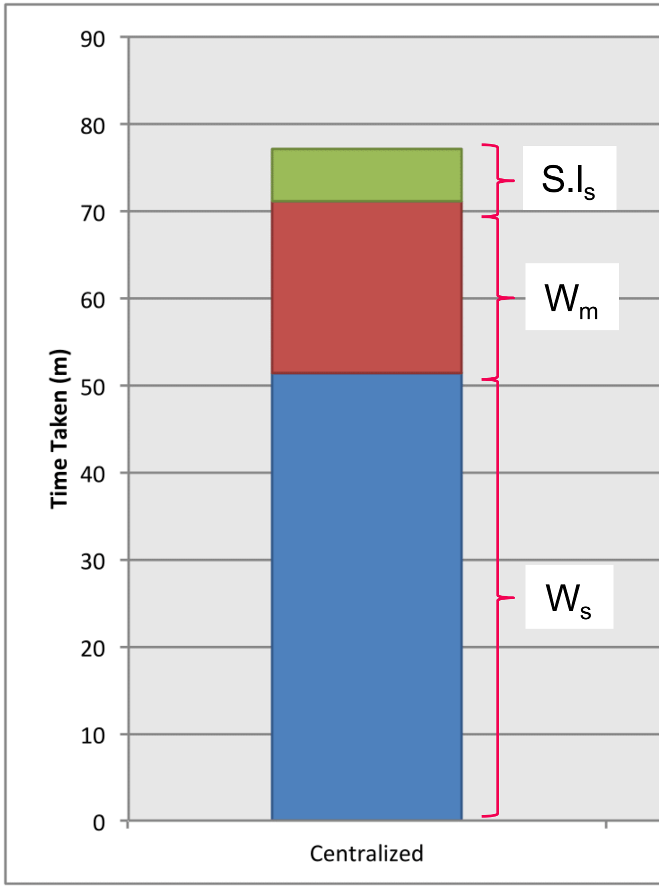


Results

Runtime behavior

- $T_{total} = W_m + W_s + S.l_s$

- Time taken while processing 100K EVs for 400 supersteps using 50 tasks.
  - 65% spent on EV optimization (W_s)
  - 25% spent on Master processing (W_m)
  - 10% spent on synchronization ($S.l_s$)



W_s can be decreased by adding more computation power.

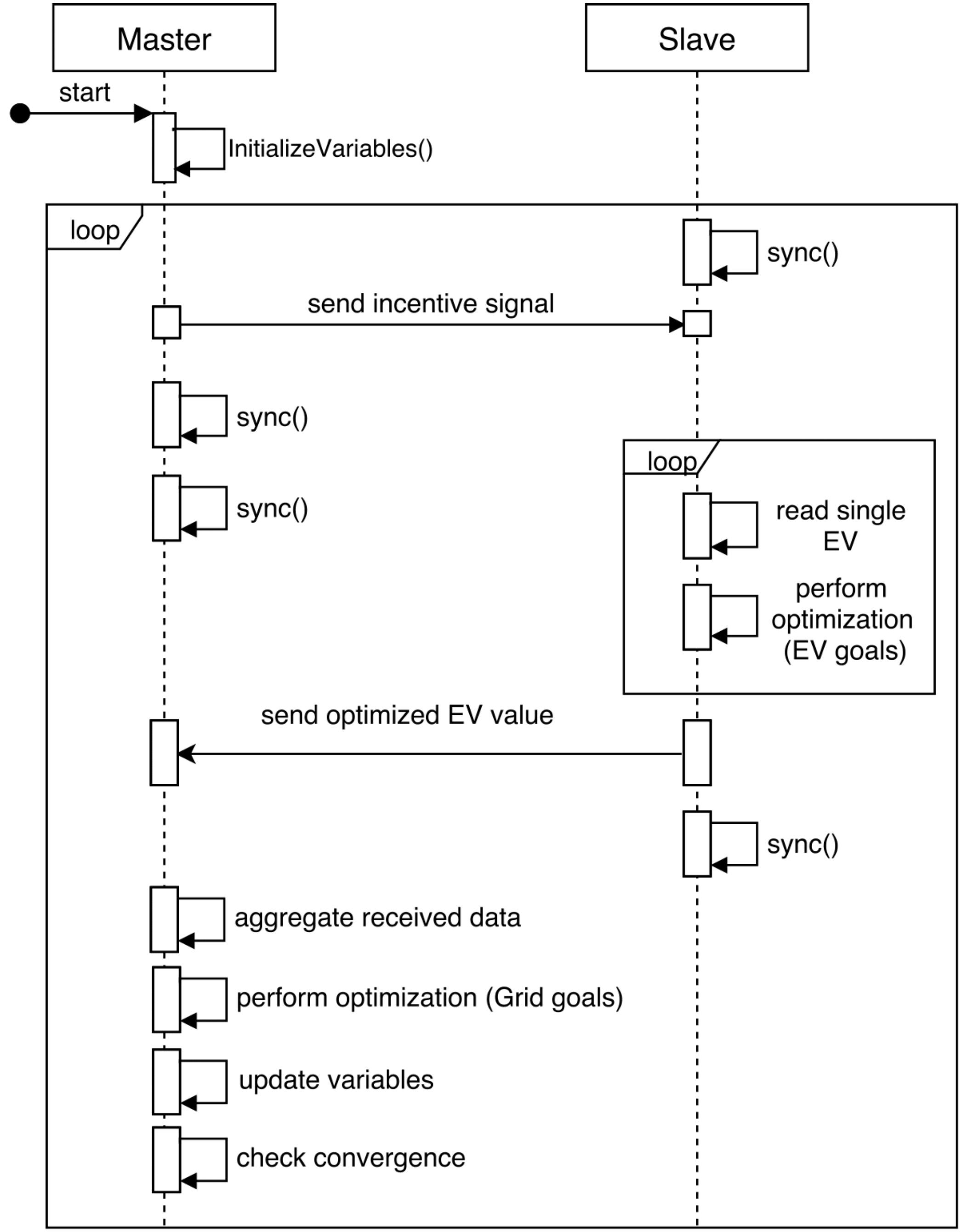
Problem: Processing on Master is a bottleneck.

Solution: Decentralized Approach

Algorithm

▷ Measured the time taken by different operations on Master

▷ Updated algorithm sends aggregated values only

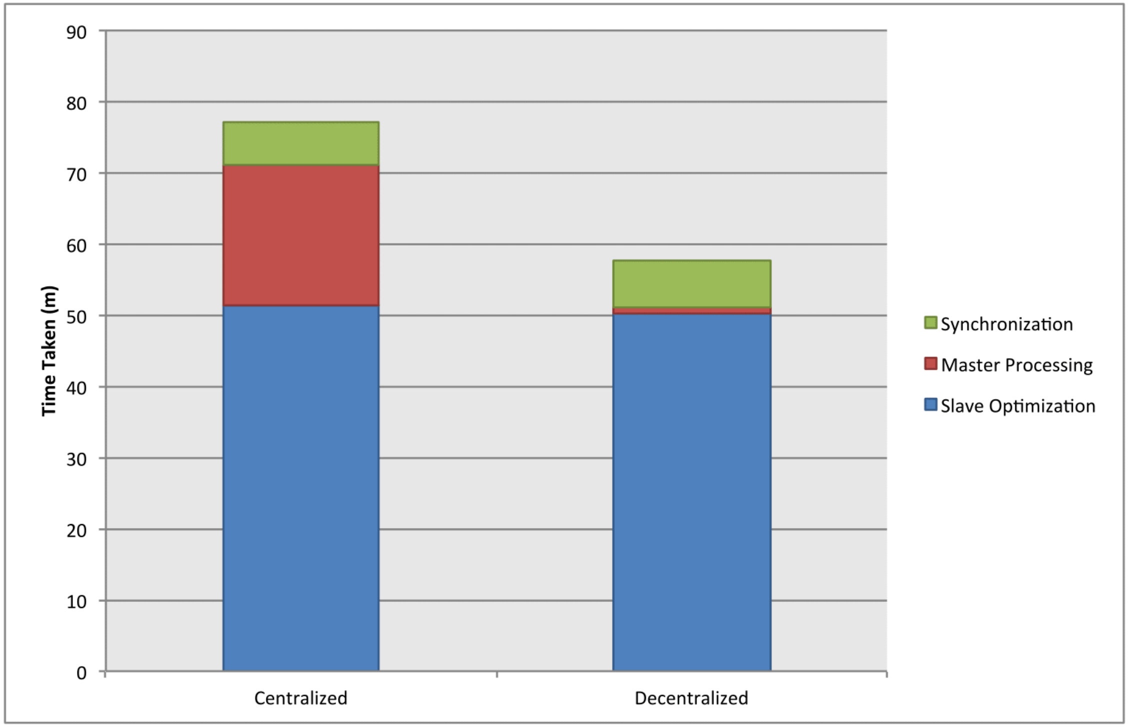


Decentralized Approach

▷ Total time reduced to 58 minutes from 78.

- 89% spent on EV optimization (W_s)
- 1% spent on Master processing (W_m)
- 10% spent on synchronization ($S.l_s$)

$$T_{total} = W_s + S.l_s$$



Runtime model

▷ $T_{total} = W_s + S.l_s$

▷ Can be reformulated in terms of total supersteps (S), total parallel process (P) and total EVs (N)

- t_c represents the time taken to solve a single EV optimization problem. Average value 6.5 ms.
- t_{sync} average time slave spends waiting for barrier synchronization. Average value 57 ms.

$$T_{total} = \frac{S.N}{P} t_c + S.P.t_{sync}$$

Total time to process 1 Mio EVs (N) in 200 supersteps (S) using 100 parallel processes (P) ?

$$T_{total} = \frac{S.N}{P} t_c + S.P.t_{sync}$$

$$T_{total} = 240 \; min^*$$

* Average estimated time

“

Total machines required to process 1 Mio EVs (N) in 100 supersteps (S) in 60 minutes (T) ?

$$M = \frac{S.N}{T_{total}.5} t_c$$

$$M = 36^*$$

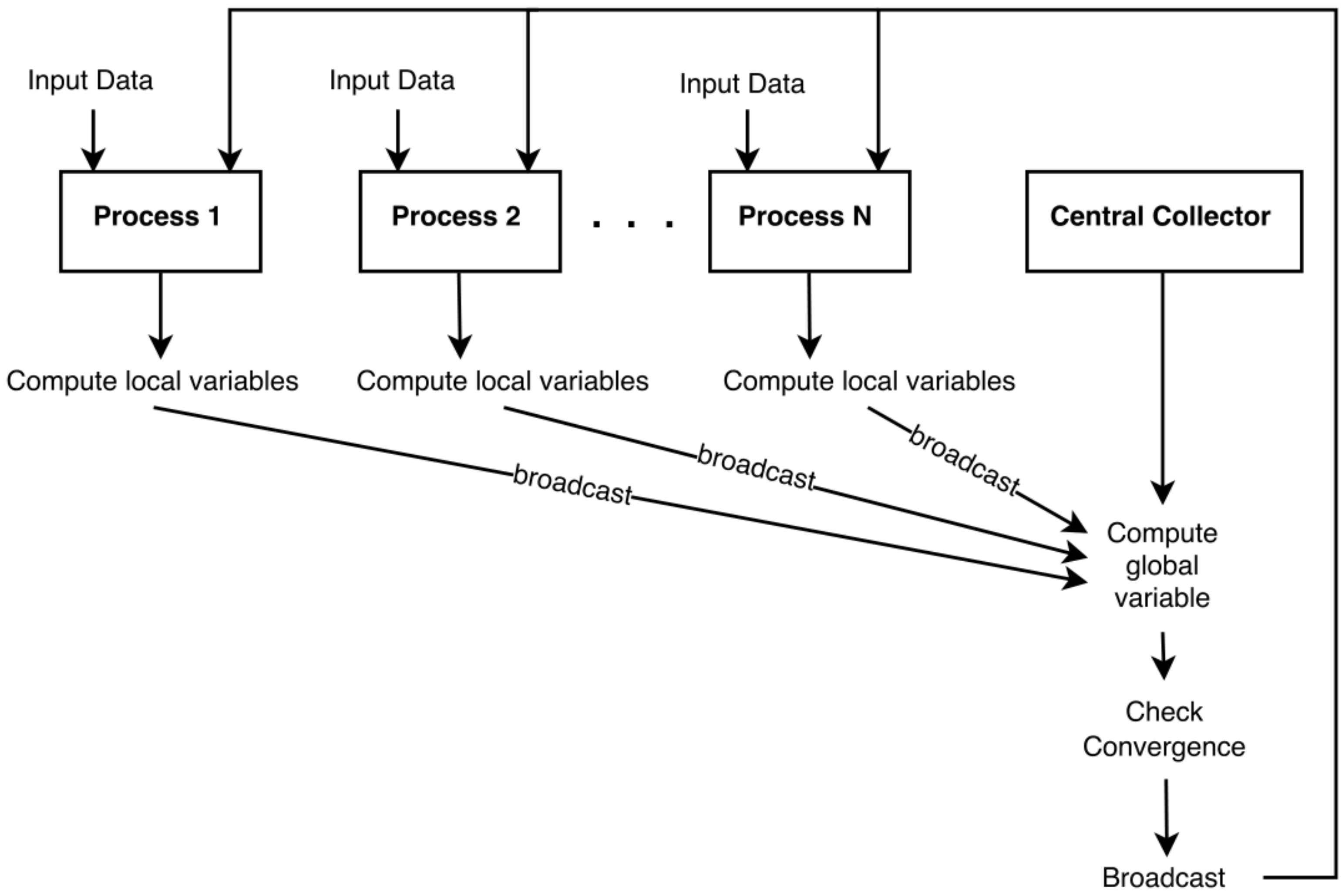
* Assuming that each machine can run 5 Hama processes in parallel.

Framework

Framework (ADF)

- ADMM Distributed Framework
- For new algorithms, implementation needs to be started from scratch
- Reusable set of classes
- Automating the optimization process as much as possible

General Idea



Requirements

- Solver agnostic
- Modeling language
- Supports multiple categories of ADMM problems
- Functions
- Abstraction on distributed system

Solver & Modeling Language Agnostic

- CPLEX, Gurobi, OPL, CPL
- Exchange problem

$$x_i^{k+1} := argmin_{x_i}(f_i(x_i) + (\rho/2)||x_i - x_i^k + \overline{x}^k + u^k||_2^2)$$
$$u^{k+1} := u^k + \overline{x}^{k+1}$$

- All we care about is the value of x_i^{k+1}

$$x_i^{k+1} := XUpdate(function\_input, local\_variables)$$

- User can implement XUpdate interface using any solver or modeling language !

Multiple ADMM Categories

- Exchange problem

$$x_i^{k+1} := argmin_{x_i}(f_i(x_i) + (\rho/2)||x_i - x_i^k + \overline{x}^k + u^k||_2^2)$$
$$u^{k+1} := u^k + \overline{x}^{k+1}$$

- Consensus problem

$$u_i := u_i + x_i - z$$
$$x_i := argmin(f_i(x_i) + (\rho/2)||x_i - z + u_i||_2^2)$$
$$z := prox_{g,N_\rho}(\overline{x} + \overline{u})$$

- Extend BSPBase, ContextBase abstract classes

Functions

- Exchange problem

$$x_i^{k+1} := argmin_{x_i}(f_i(x_i) + (\rho/2)||x_i - x_i^k + \overline{x}^k + u^k||_2^2)$$
$$u^{k+1} := u^k + \overline{x}^{k+1}$$

- Implement XUpdate interface for new functions and provide them at startup

Abstraction on distributed system

```
ADFJob job = new ADFJob();

job.setMaxIteration(4);
job.setJobName("ADF Exchage EVADMM job");
job.setInputPath("aggregator.txt,EVs.txt");
job.setOutputPath("output/");
job.setSolutionVectorSize(96);

job.setADMMClass(BSPExchange.class);
job.setFunction1(ValleyFillingOptimizationFunction.class);
job.setFunction2(EVOptimizationFunction.class);

job.run();
```

Conclusion

Conclusion

- ▷ Reproduce EVADMM on distributed environment
  - Data aggregation on slaves massively decreases the total runtime. Parallelize time consuming serial parts of algorithm.
- ▷ Foundation for a general framework to solve similar problems
- ▷ Spark vs Hama -> Go with Spark
  - Performance is equal
  - Hama has very small community
  - Even better -> Use Apache Beam/Google DataFlow

Conclusion

- Reproduce EVADMM on distributed environment
  - Data aggregation on slaves massively decreases the total runtime. Parallelize time consuming serial parts of algorithm.
- Foundation for a general framework to solve similar problems
- Spark vs Hama -> Go with Spark
  - Performance is equal
  - Hama has very small community
  - Even better -> Use Apache Beam/Google DataFlow
- Contributed to Apache Hama to add round-robin based task allocation.

Thanks!

Any questions?

You can find me at:
behroz.sikander@tum.de

References

▷ S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein. "Distributed optimization and statistical learning via the alternating direction method of multipliers."

▷ S. Boyd and L. Vandenberghe. "Convex programming."

▷ J. Rivera, P. Wolfrum, S. Hirche, C. Goebel, and H.-A. Jacobsen. "Alternating direction method of multipliers for decentralized electric vehicle charging control."

▷ S. Seo, E. J. Yoon, J. Kim, S. Jin, J.-S. Kim, and S. Maeng. "Hama: An efficient matrix computation with the mapreduce framework."

▷ L. G. Valiant. "A bridging model for parallel computation."

Backup Slides

Hama vs Spark

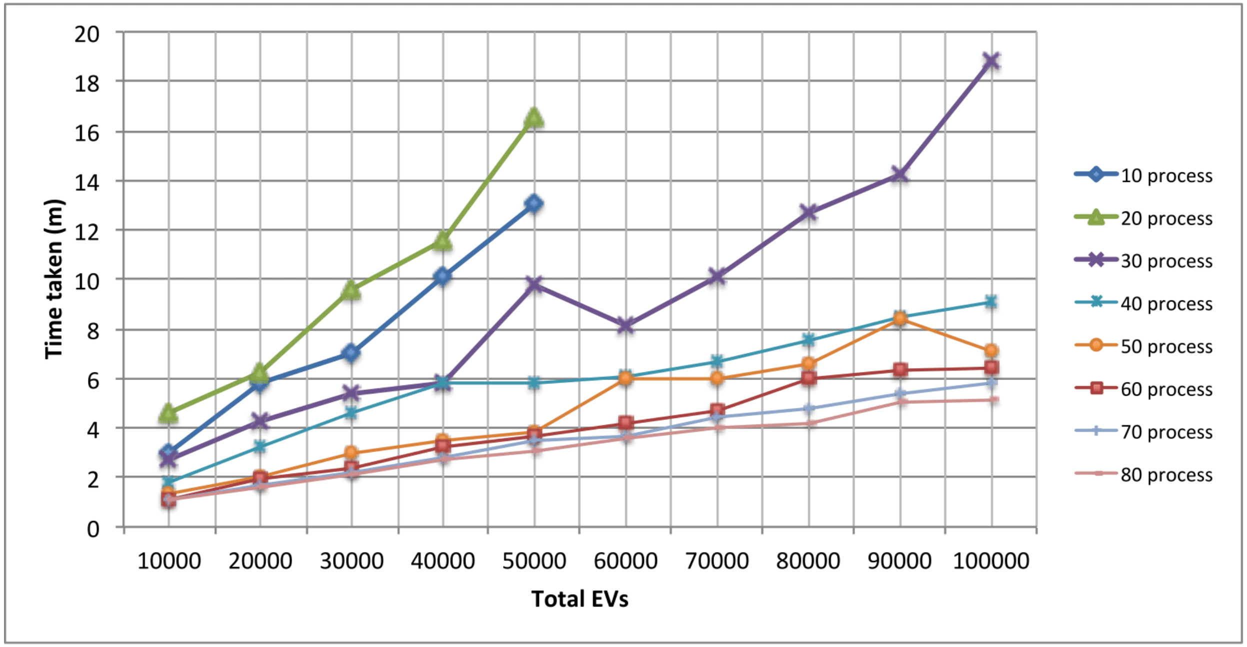
- ▷ From usability perspective, Spark is better than Hama
  - No need to understand the complications of distributed system
- ▷ Hama provides better control on communication and synchronization than Spark
- ▷ Hama is based on strong theoretical backgrounds whereas Spark is still evolving
- ▷ Performance wise, both are more or less equal
  - Recent paper shows that Hama is actually better when performing joins on big datasets
- ▷ Further, ADMM is more natural to implement in BSP.

Lessons

- ▷ Avoid loading data from file system -> In memory
- ▷ Measure load on CPUs early. Overloaded CPU decreases performance
- ▷ Parallel part should be very optimized and carefully implemented. -> Use profiling
- ▷ 3^{rd} party -> Memory leaks

Synchronization Behavior

- More data -> increase in synchronization time !
- Reason: Irregular processing time of slaves
- More processing power -> decreased synchronization time
- ~10-20% increase in sync. time for adding 10K EVs.



Centralized Master Processing Behavior

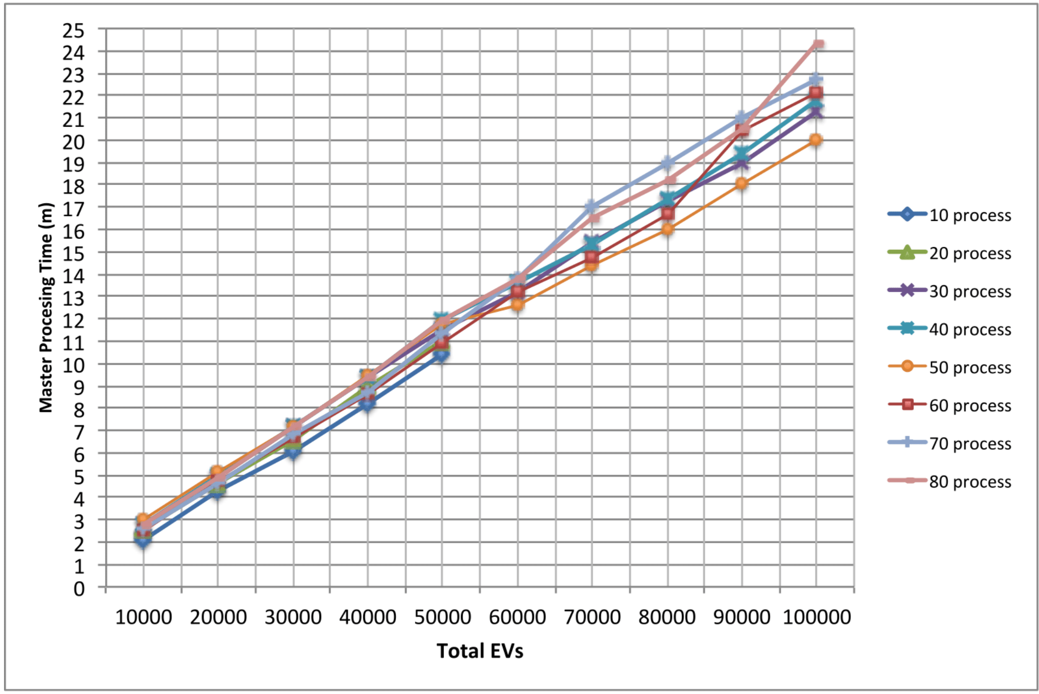
▷ Time taken by master to process in 400 supersteps

- x-axis shows total EVs, y-axis shows time taken in minutes
- Each line represents a specific number of parallel tasks used

▷ More data means to more time process

▷ Increasing computation power has no effect. Reason: 1 master !

▷ For 100,000 EVs in 400 supersteps it takes 25 minutes !!



Decentralized Master Processing Behavior

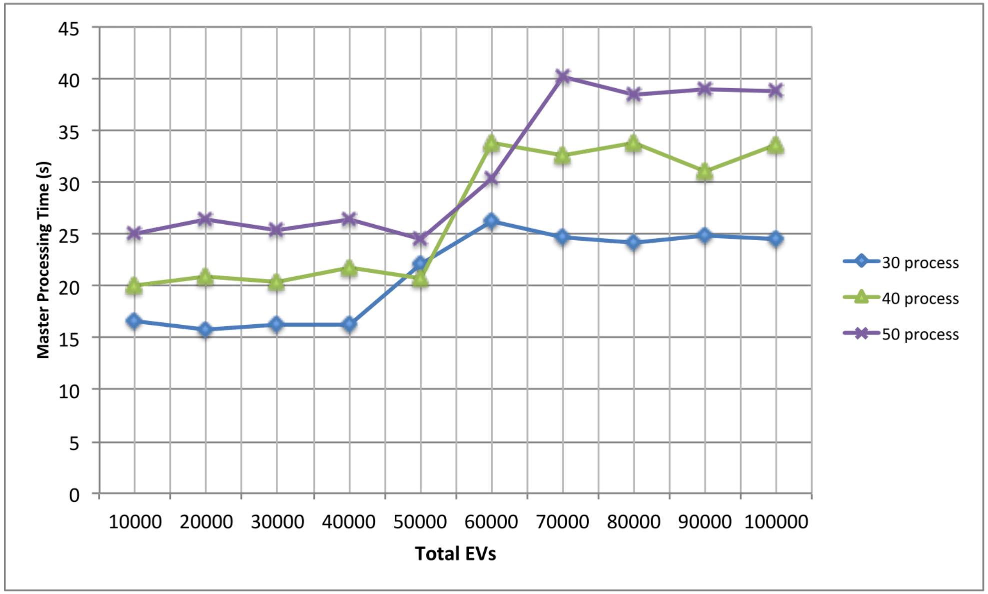
▷ Time taken by master to process in 400 supersteps

- x-axis shows total EVs, y-axis shows time taken in minutes
- Each line represents a specific number of parallel tasks used

▷ No change while increasing data or processing power.

▷ Reason: Number of incoming messages to process stays the same !

▷ For 100,000 EVs in 400 supersteps it takes 40 seconds!!



Why not 1 Mio ?

- Unreasonable time taken to process with current cluster

General Idea

▷ Exchange problem

- x-update can be done independently (local variable)
- u-update needs to be computed globally because it depends on all x-updates. (global variable)

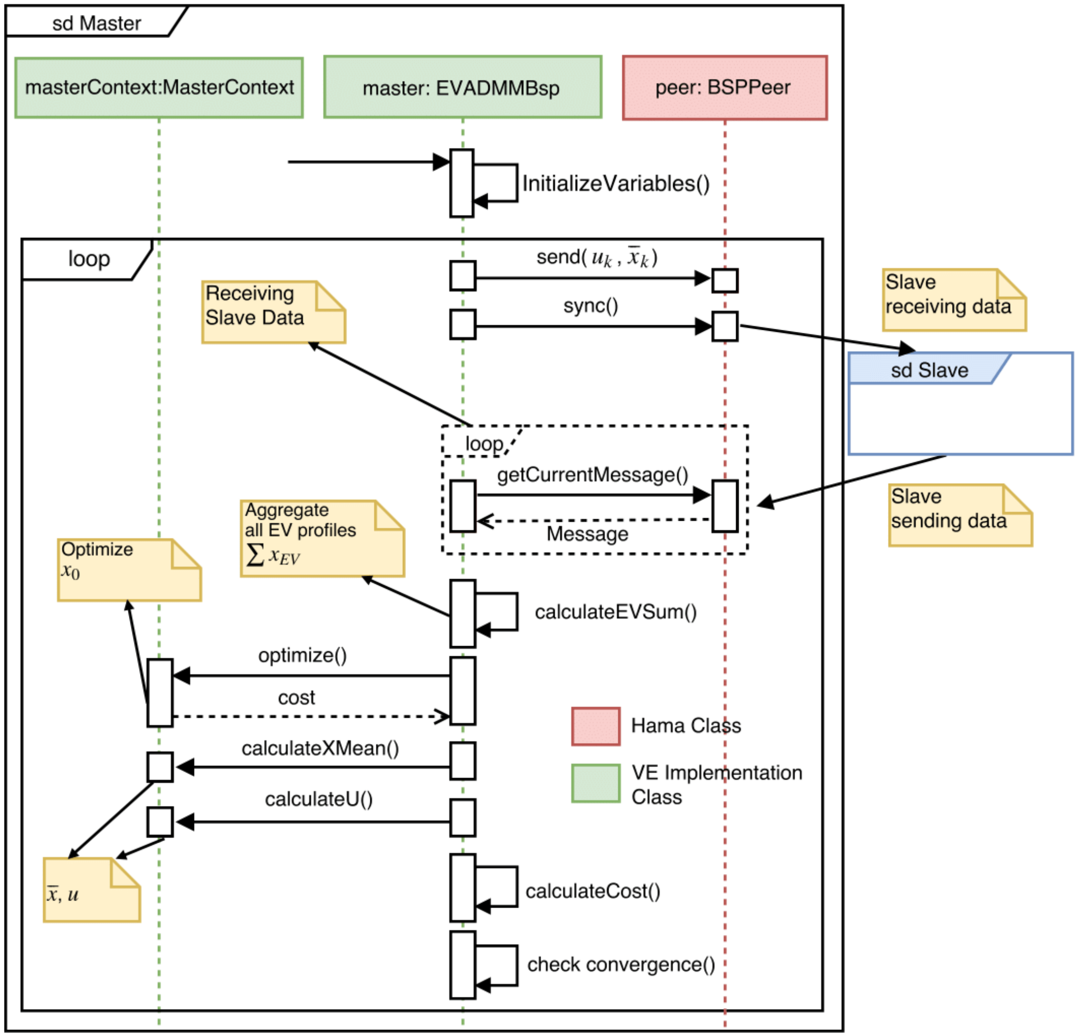
$$x_i^{k+1} := argmin_{x_i}(f_i(x_i) + (\rho/2)||x_i - x_i^k + \overline{x}^k + u^k||_2^2)$$
$$u^{k+1} := u^k + \overline{x}^{k+1}$$

▷ Consensus problem

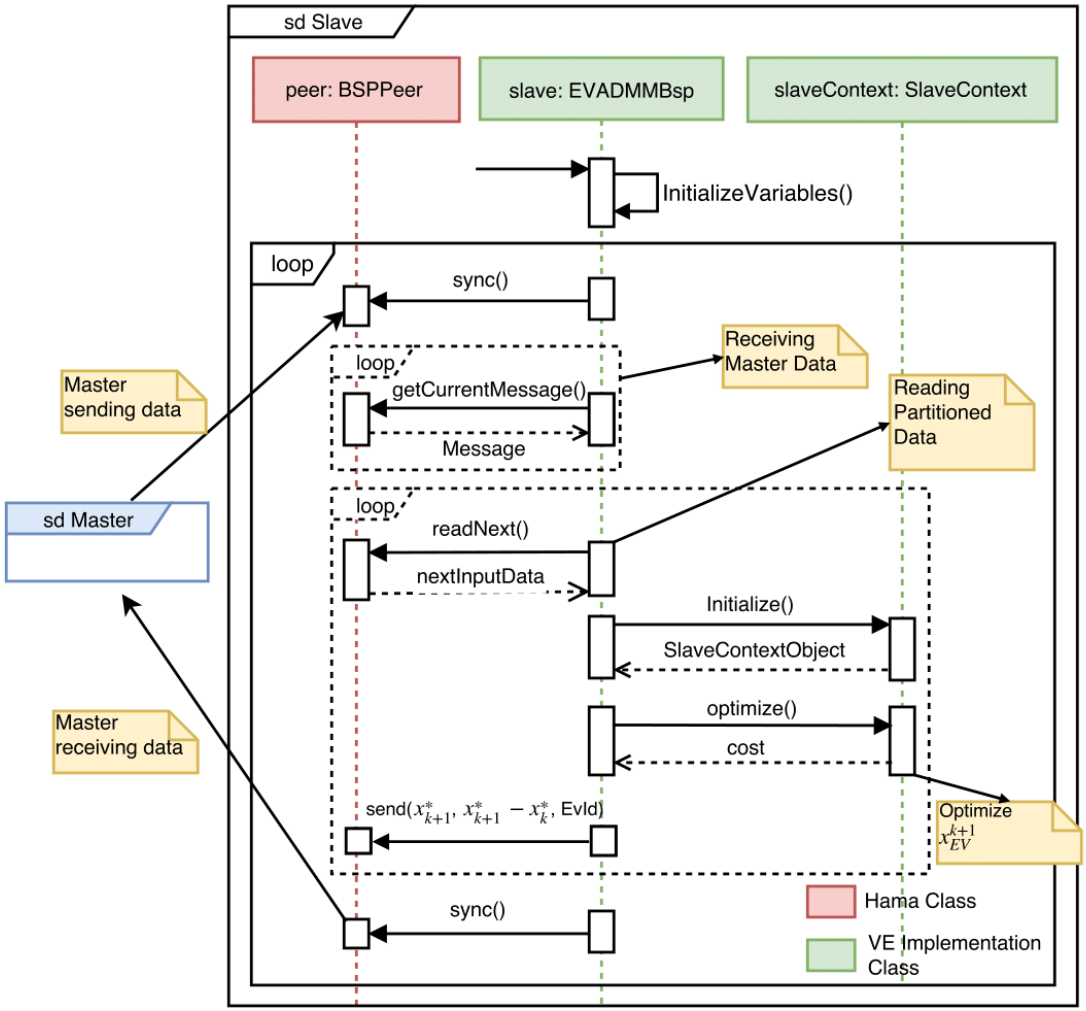
- u and x update can be carried out independently
- z-update can only be computed globally

$$u_i := u_i + x_i - z$$
$$x_i := argmin(f_i(x_i) + (\rho/2)||x_i - z + u_i||_2^2)$$
$$z := prox_{g,N_\rho}(\overline{x} + \overline{u})$$

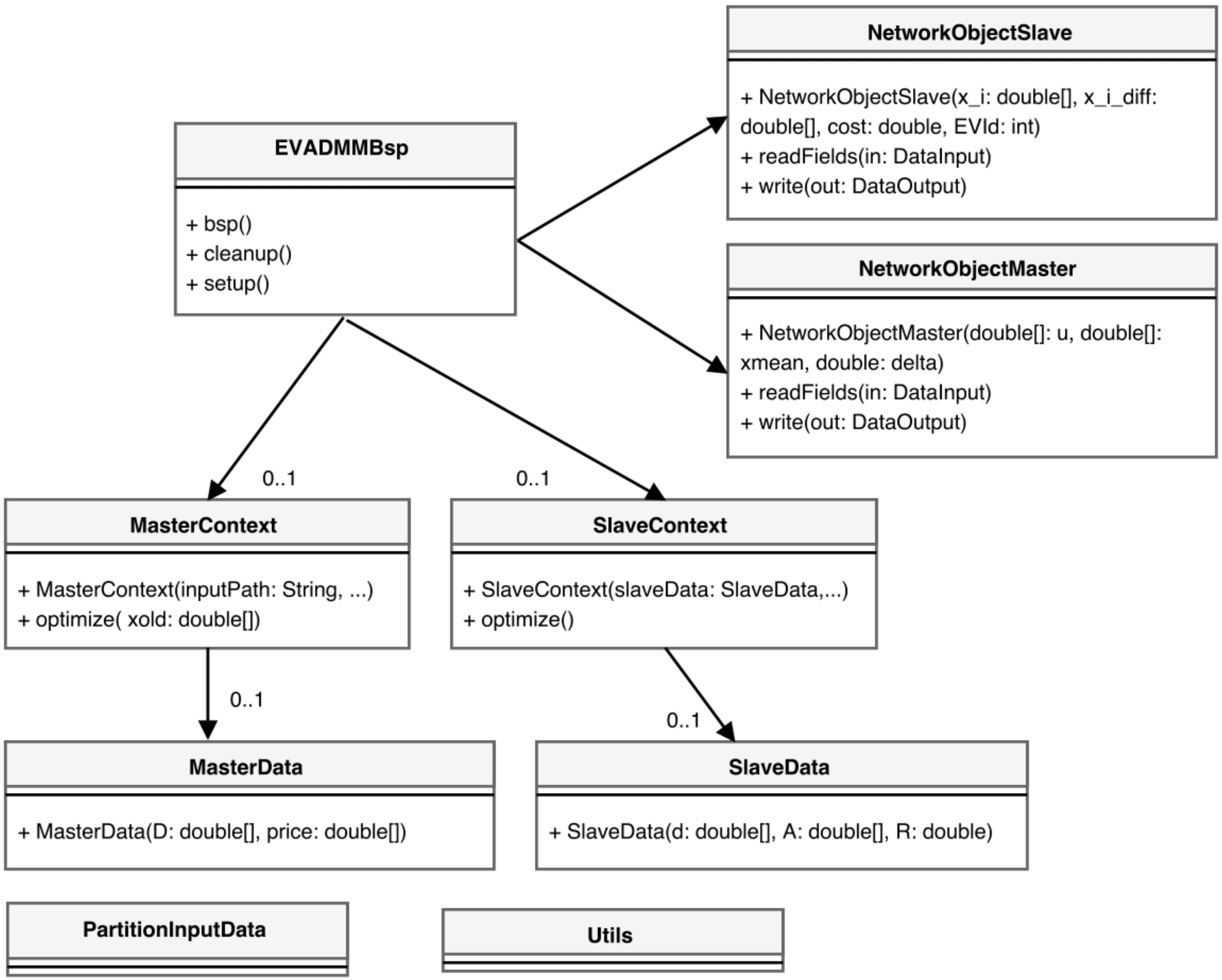
VE- Sequence Diagram



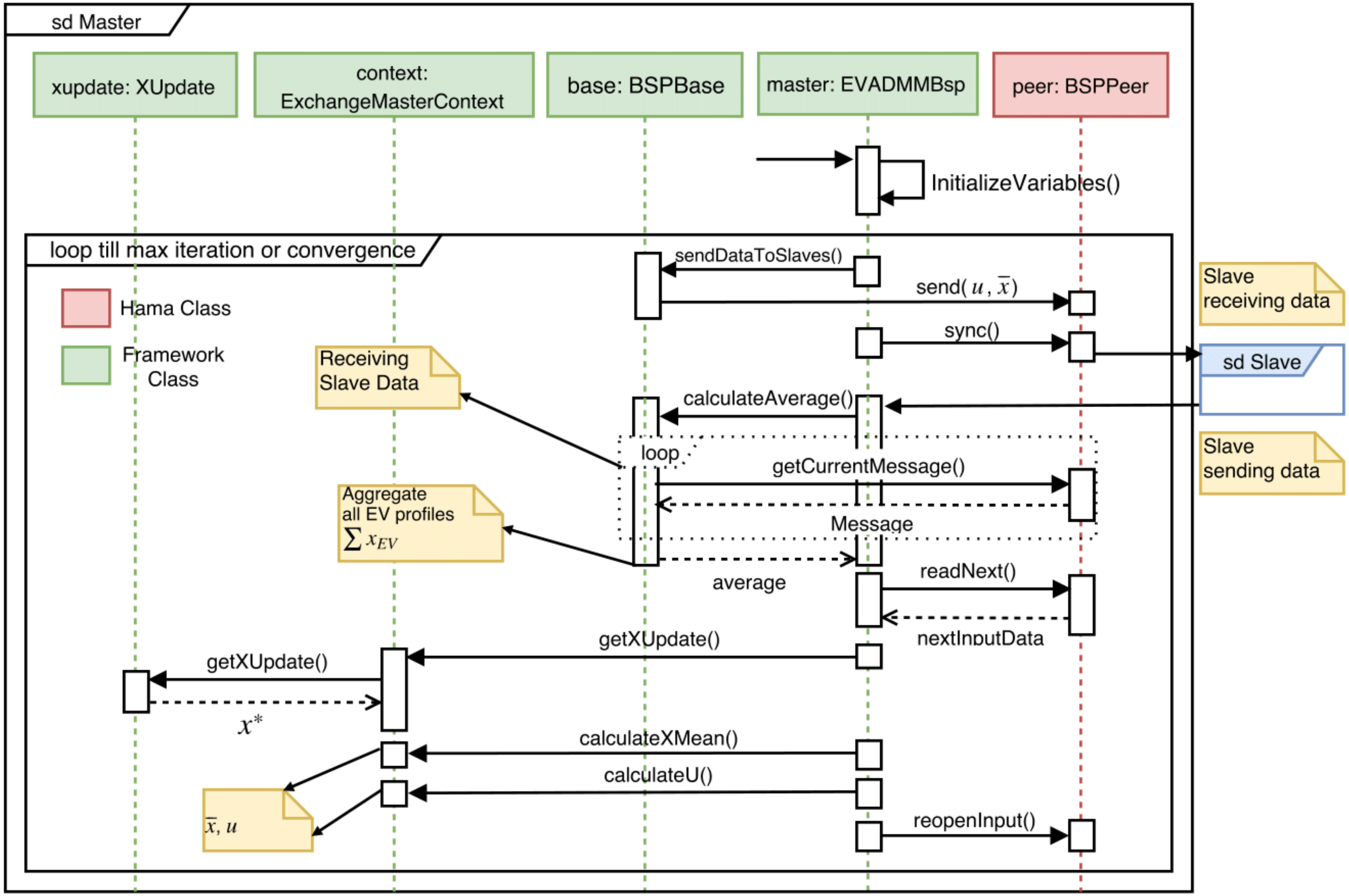
VE- Sequence Diagram



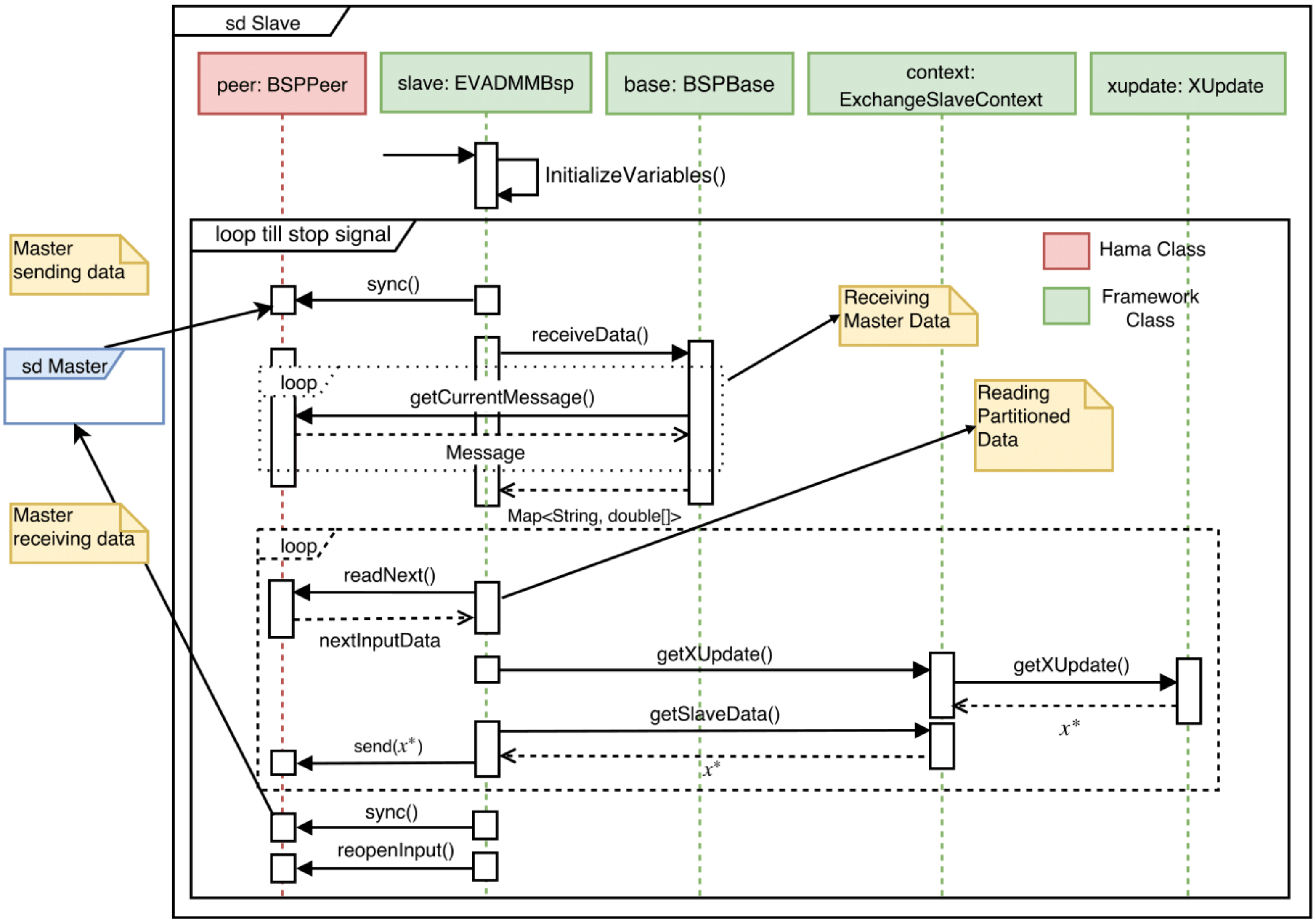
VE- Class Diagram



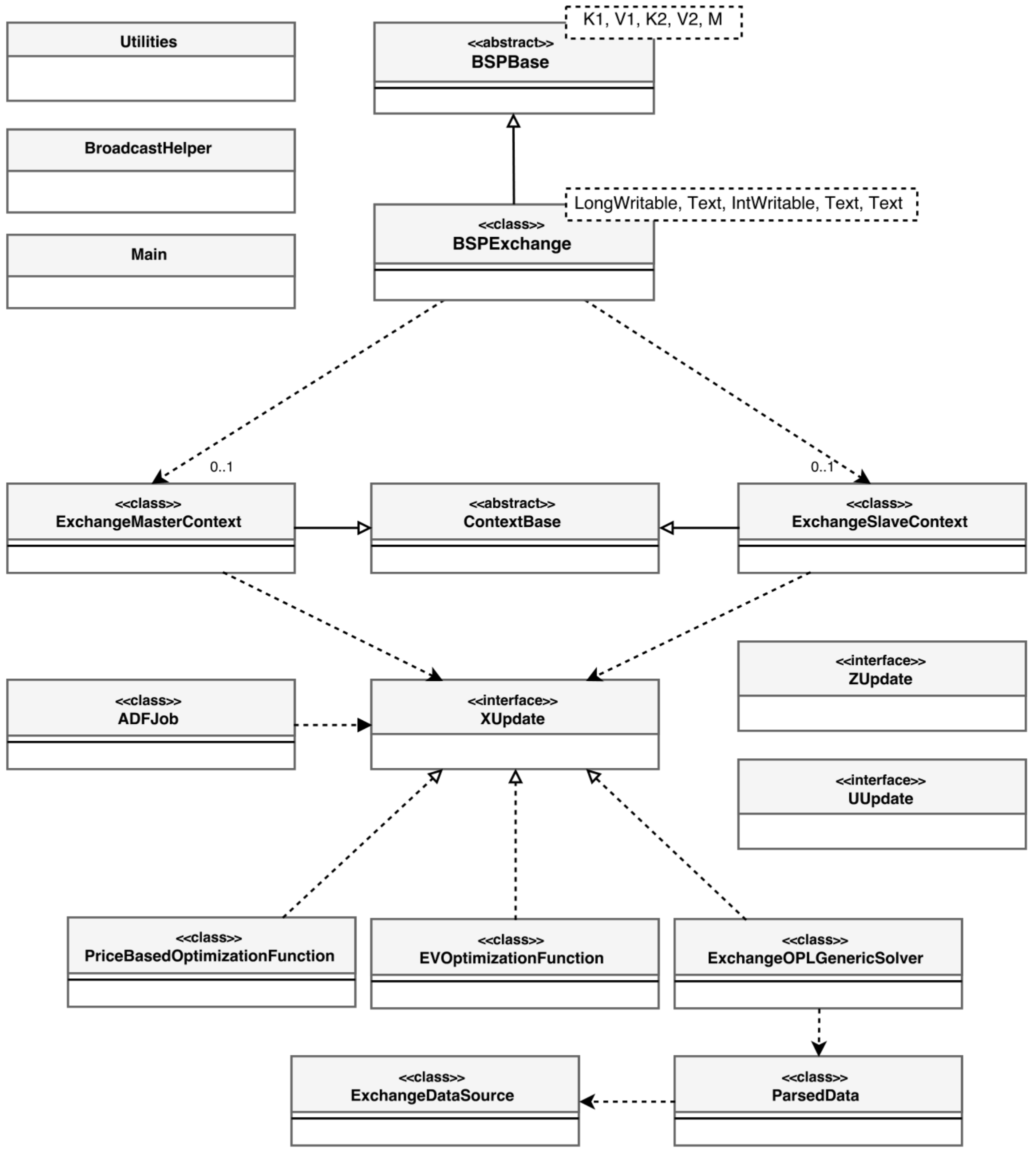
Framework- Sequence Diagram



Framework- Sequence Diagram



Framework- Class Diagram



Background

Optimization

- ▷ ***Best solution*** from a set of alternatives
- ▷ while being ***constrained by a criteria***
- ▷ Examples, portfolio optimization, device sizing

$$
\begin{array}{ll}
\text{minimize} & f_0(x) \\
\text{subject to} & f_i(x), i = 1, \ldots, m
\end{array}
$$

Convex Optimization

- Objective and constraint functions as ***convex*** or ***concave***
- ***Solution guaranteed !***

ADMM

- Alternating Direction Method of Multipliers
- Used for *distributed* convex optimization
- Converts problems to local sub-problems
- Use local solutions to find global solution
- Iterative

$$\begin{array}{ll} \text{minimize} & f(x) + g(z) \\ \text{subject to} & Ax + Bz = c \end{array}$$

ADMM

- Distributed ADMM form

$$x^{k+1} := argmin_x(f(x) + (\rho/2)||Ax + Bz^k - c + u^k||_2^2)$$
$$z^{k+1} := argmin_z(g(z) + (\rho/2)||Ax^{k+1} + Bz - c + u^k||_2^2)$$
$$u^{k+1} := u^k + Ax^{k+1} + Bz^{k+1} - c$$

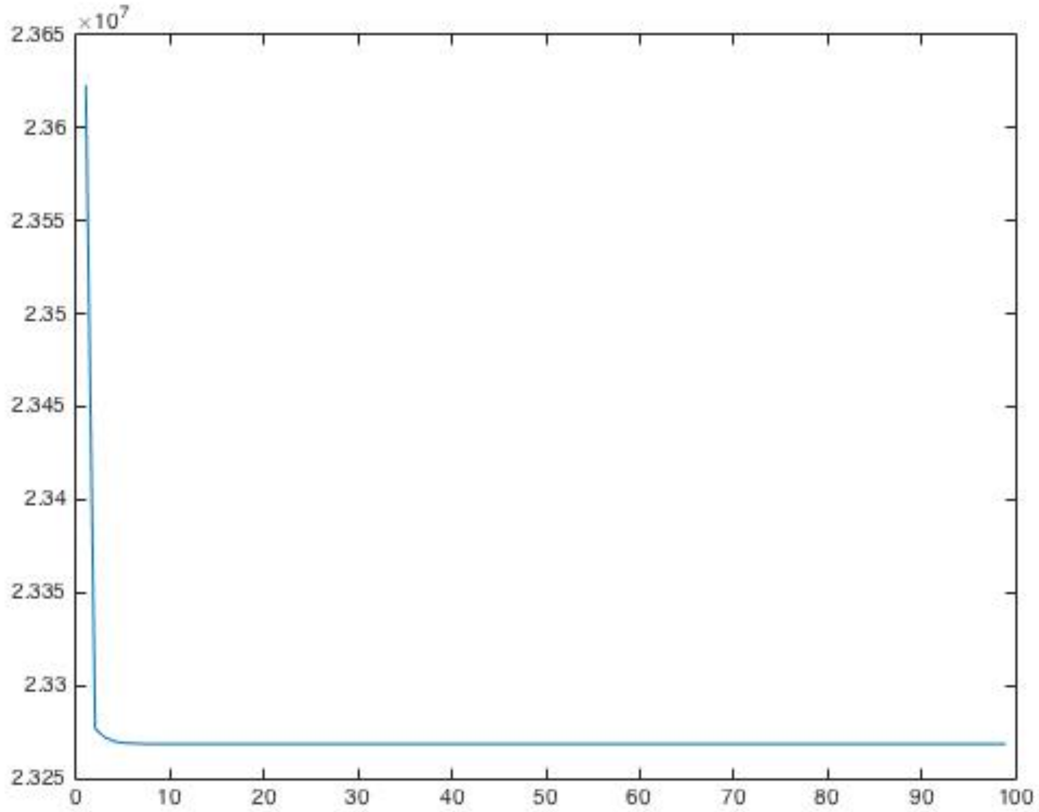
Solvers

- Software *tools* to solve mathematical problems.
- *Abstract out* all the complexities
- Simple programming interface
- *Cost functions and constraints can be modeled*
- Tools: CPLEX, Gurobi, SCIP, CLP, LP_SOLVE
- This thesis : *CPLEX*

EVADMM

- Increasing carbon emissions
- Vehicles are a big source
- Solution: Electric Vehicles (EV)
- Charging an EV takes more power than a normal house
- Solution: Increase infrastructure
- Or controlled charging

Convergence



Solvers

$$x_i^{k+1} =$$
$$\begin{aligned} \text{minimize} \quad & \rho/2 ||x_i - x_i^k + \bar{x}^k + u^k||_2^2 \\ \text{subject to} \quad & A_i x_i = R_i \\ & \underline{x}_i \leq x_i \leq \overline{x}_i \end{aligned}$$

```java
public double optimize(IloCplex cplex){
    cplex.clearModel();
    cplex.setOut(null);
    IloNumVar[] x_i = new IloNumVar[this.x.length];
    IloNumExpr[] exps = new IloNumExpr[x.length];
    IloNumExpr[] AXExpEq = new IloNumExpr[x.length];

    double[] data = subtractOldMeanU(this.x);

    for (int i = 0; i < this.x.length; i++) {
        x_i[i] = cplex.numVar(xi_min[i], xi_max[i]);
        exps[i] = cplex.prod(rho / 2, cplex.square(cplex.sum(x_i[i],
                                    cplex.constant(data[i]))));
        AXExpEq[i] = cplex.prod(x_i[i], this.slaveData.getA()[i]);
    }
    cplex.addMinimize(cplex.sum(exps));
    cplex.addEq(cplex.sum(AXExpEq), this.slaveData.getR());

    if (cplex.solve()) {
        x_optimal = cplex.getValues(x_i);
    }
}
```

Modeling Language

- Specify equations more *naturally*
- Internally convert to solver understandable format
- Model + data file as input

Modeling Language

$$x_i^{k+1} = \begin{aligned} &\text{minimize} \;\; \rho/2||x_i - x_i^k + \bar{x}^k + u^k||_2^2 \\ &\text{subject to} \;\; A_i x_i = R_i \\ &\qquad\qquad\quad \underline{x}_i \leq x_i \leq \overline{x}_i \end{aligned}$$

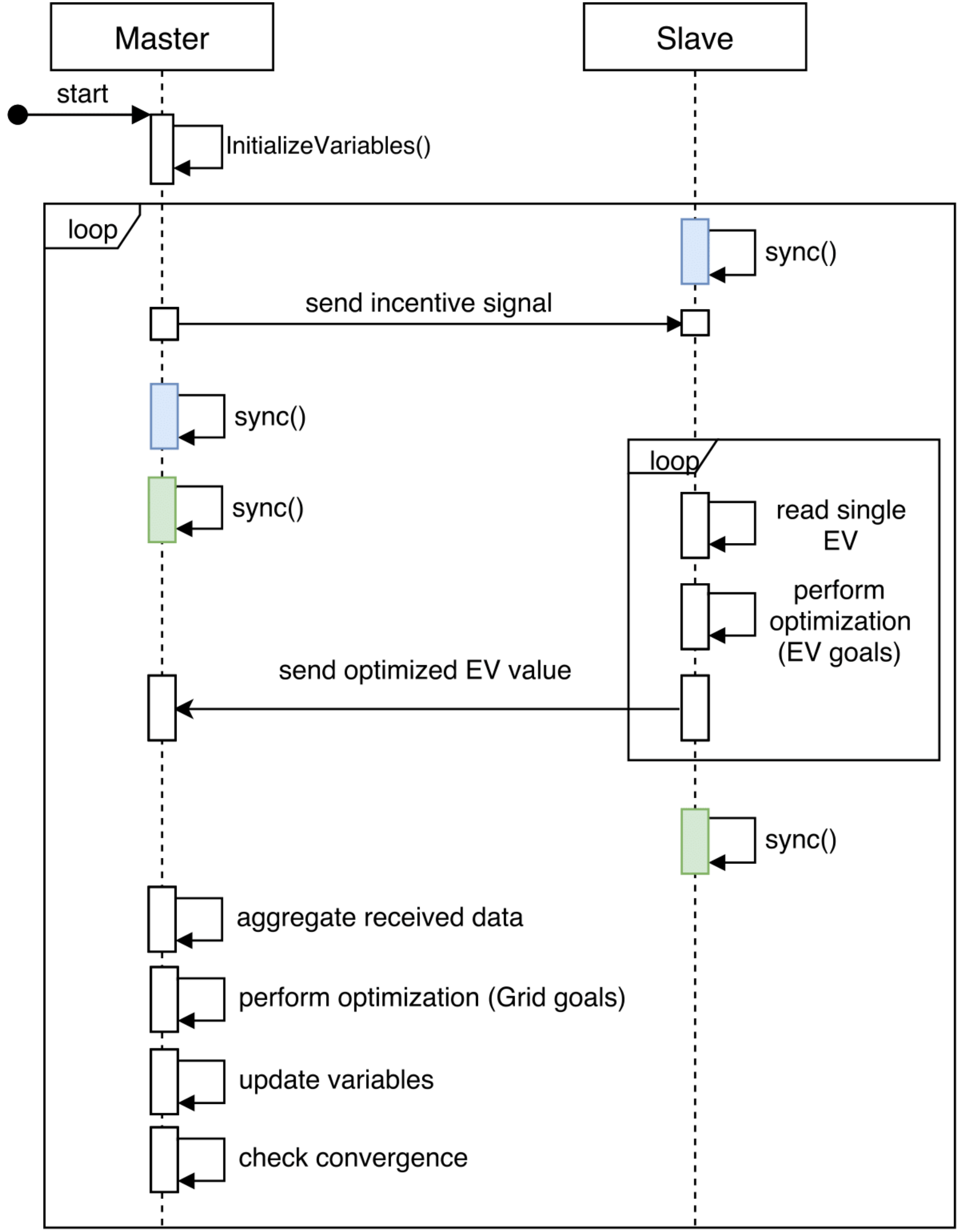
```
dvar float xOptimal[i in R] in xi_min[i]..xi_max[i];
minimize
    sum(i in R)
        (
            (rho / 2) *
                (xOptimal[i] - xOld[i] + xMean[i] + u[i]) *
                (xOptimal[i] - xOld[i] + xMean[i] + u[i])
        );
subject to {
    sum(i in R)
        xOptimal[i] * A[i] == R_value;
}
```

Algorithm

- 2 supersteps per iteration
- To process 100K EVs, all

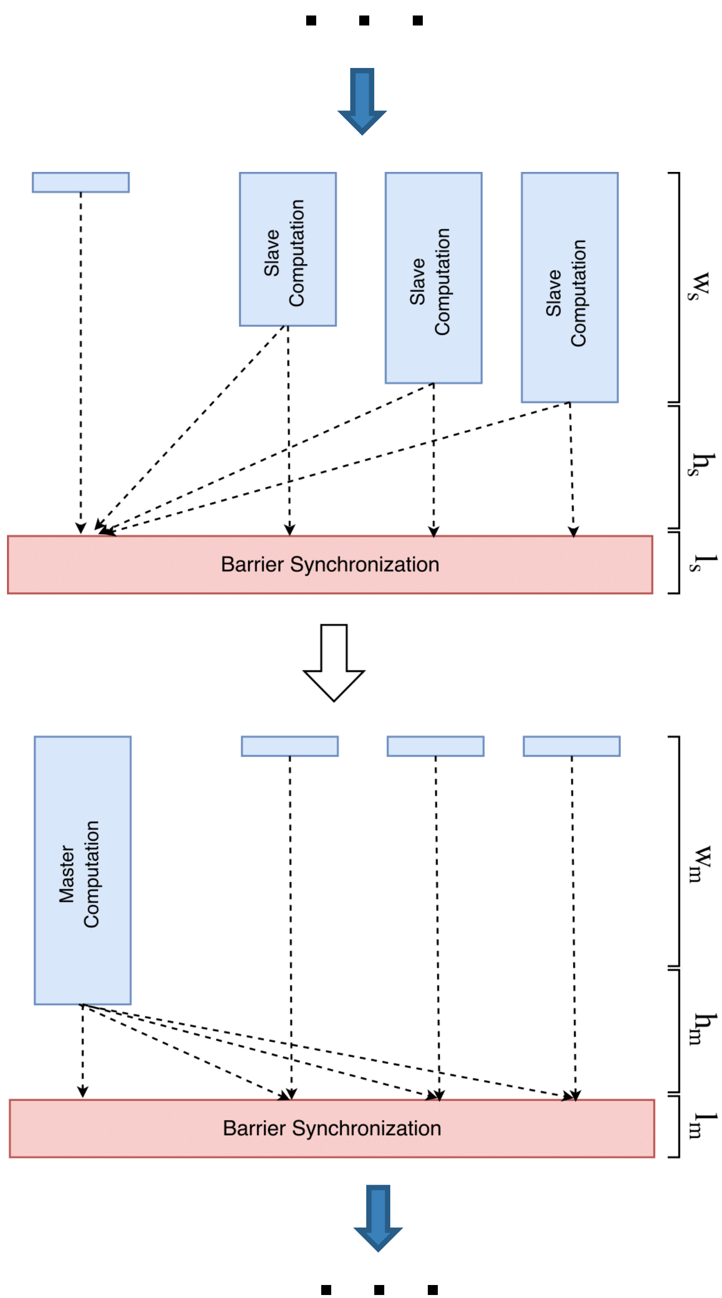
Slaves will send 100K messages

to Master



Supersteps

- ▷ 2 supersteps per iterations
- ▷ $T_{total} = T_{slave} + T_{master}$
- ▷ Master/slave communication (h_m/h_s) and synchronization time (l_m) are insignificant
- ▷ $T_{total} = W_m + W_s + S.l_s$



Deployment

Docker

- ▷ light weight, secure, open source project
- ▷ Package application and dependencies in Containers
- ▷ Easily create, deploy and run containers

- ▷ This thesis
  - Deployment on compute server
  - 3 containers
  - HDFS and Hama configured

Docker - Problems

- /etc/hosts file read only
- Dynamic Ips
- No public IP
- Single harddrive
- No network latency

Problems- LRZ

- Memory leaks
- Processes randomly crashing
- Firewall issues
- Hama non-uniform process distribution

Top command

```
behroz@vm-10-155-208-44: ~
top - 23:14:05 up  4:30,  1 user,  load average: 5.31, 4.43, 3.88
Tasks: 110 total,   2 running, 108 sleeping,   0 stopped,   0 zombie
%Cpu(s):  1.1 us,  1.2 sy,  0.0 ni, 97.6 id,  0.0 wa,  0.0 hi,  0.1 si,  0.0 st
KiB Mem:  16434200 total,  3249908 used, 13184292 free,   137656 buffers
KiB Swap:  1048572 total,        0 used,  1048572 free.   216760 cached Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
28233 behroz    20   0 3685060 338640  18952 S   2.3  2.1  16:17.38 java
28205 behroz    20   0 3685060 307372  18936 S   2.0  1.9  16:24.88 java
28262 behroz    20   0 3685060 420960  18984 S   1.7  2.6  16:23.89 java
28286 behroz    20   0 3685060 326644  18896 S   1.7  2.0  16:28.86 java
28346 behroz    20   0 3685060 356076  18864 S   1.7  2.2  16:31.69 java
28319 behroz    20   0 3685060 440540  18964 S   1.3  2.7  16:30.80 java
```

Related Work

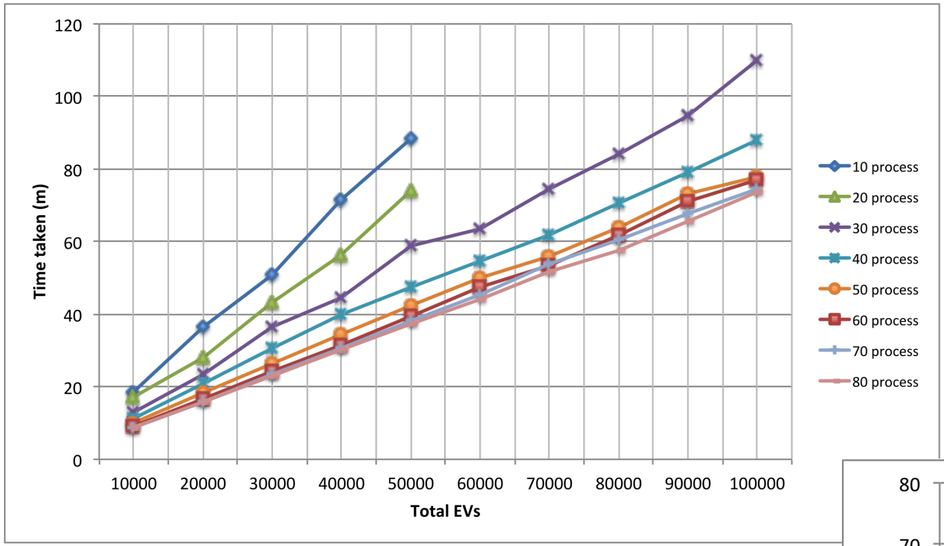
Serial Implementation

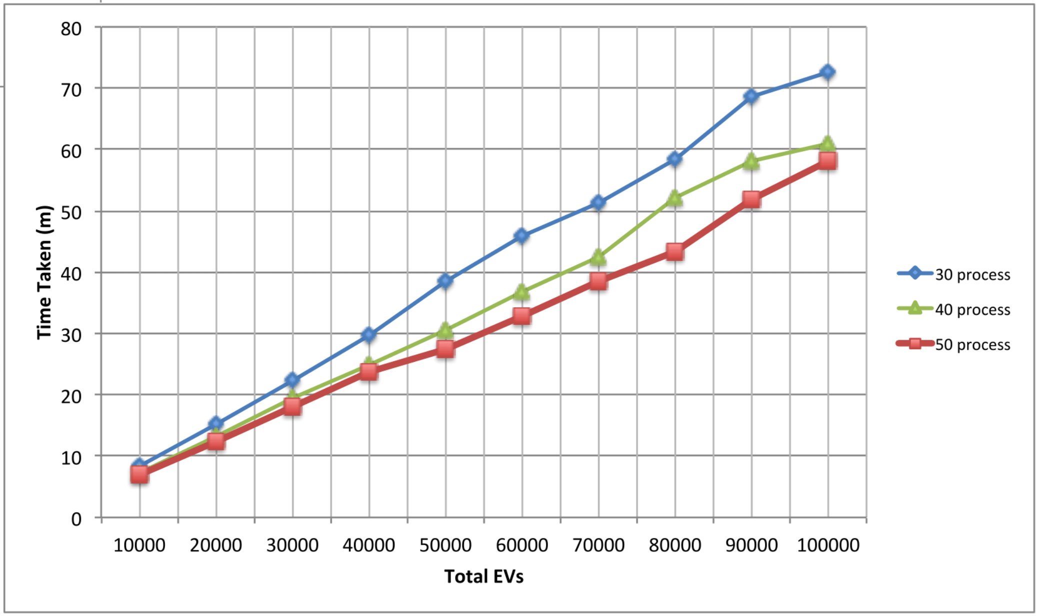
- Implemented in Matlab
- Solver: CVXGEN
- Data processed: 100 EV
- Total processing time: 2 minutes

Distributed Implementation

- Implemented in *Matlab*
- *parfor* function of Parallel Computing Toolbox used
- 1 machine with 16 cores and 64 GB RAM
- Data processed: 100,000
- Solver: CVXGEN
- Time: *100,000 EVs* processed in *30 minutes*

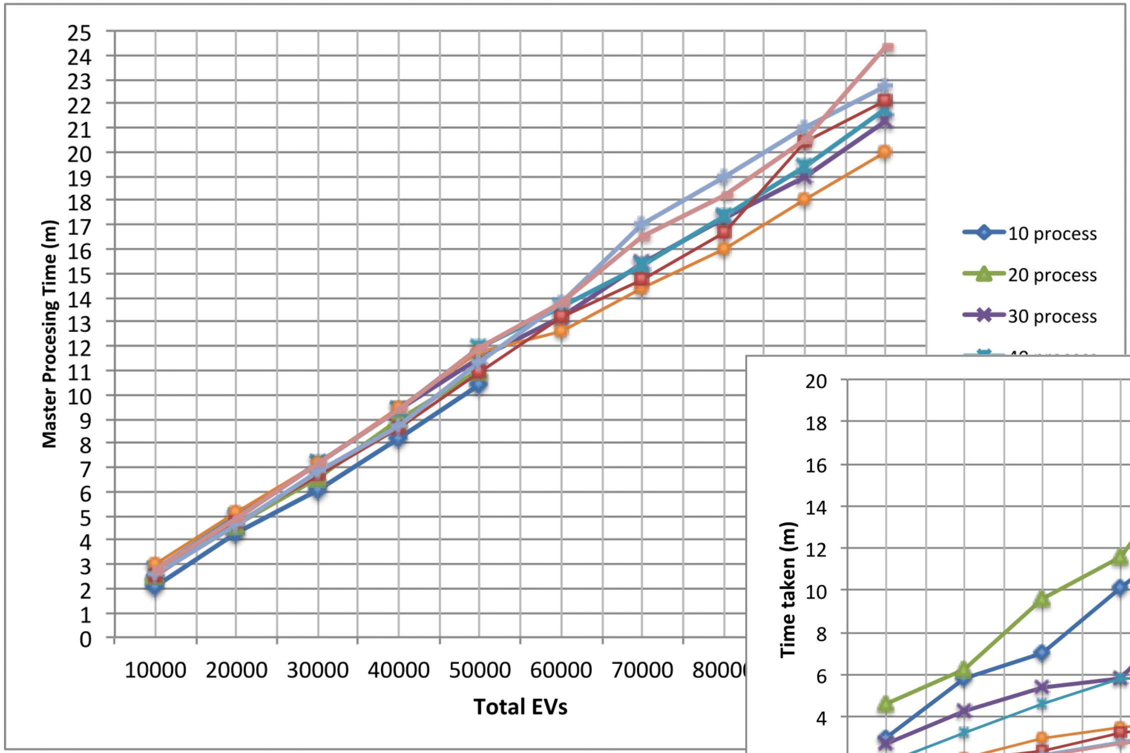
Overall runtime



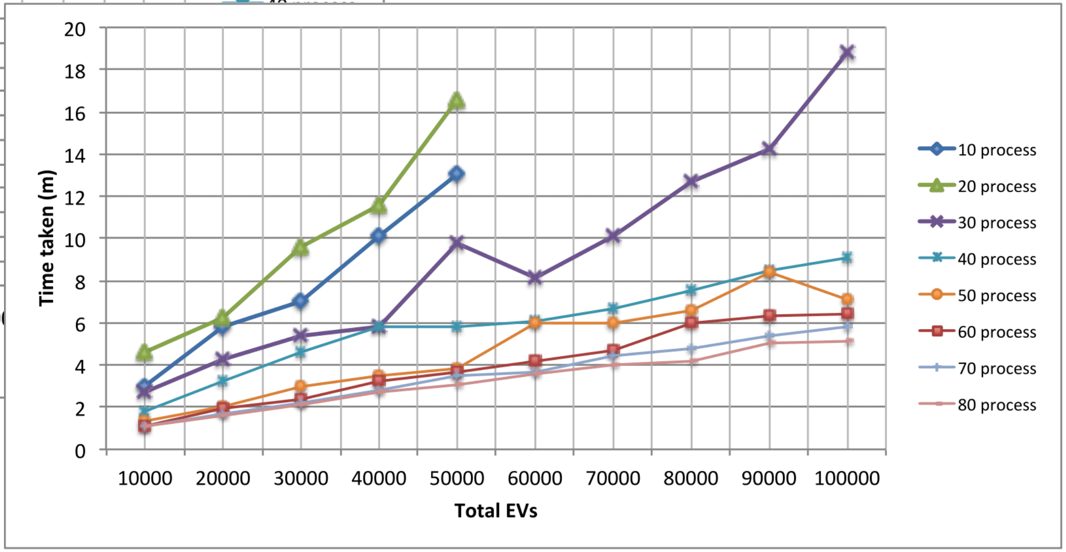


Matlab vs Hama Implementation

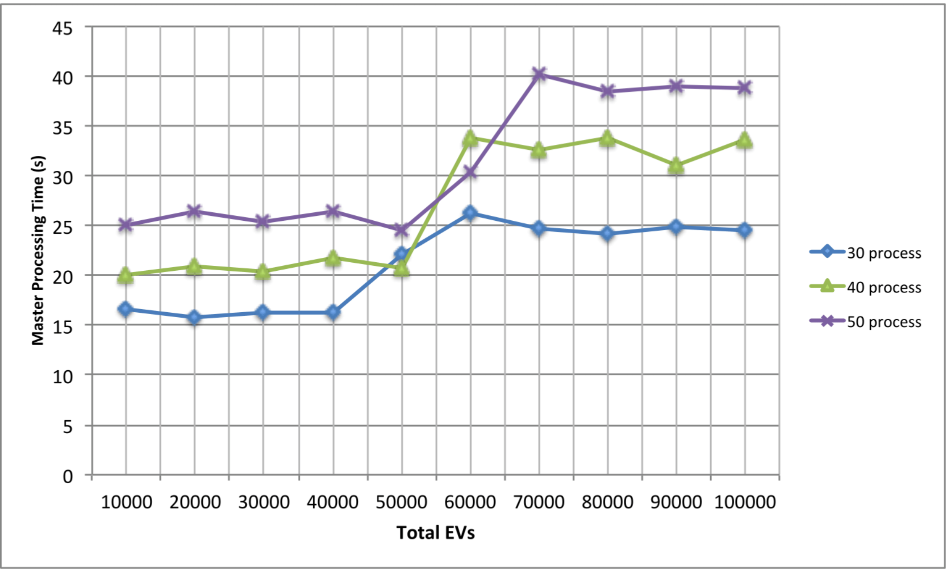
Change at runtime ?



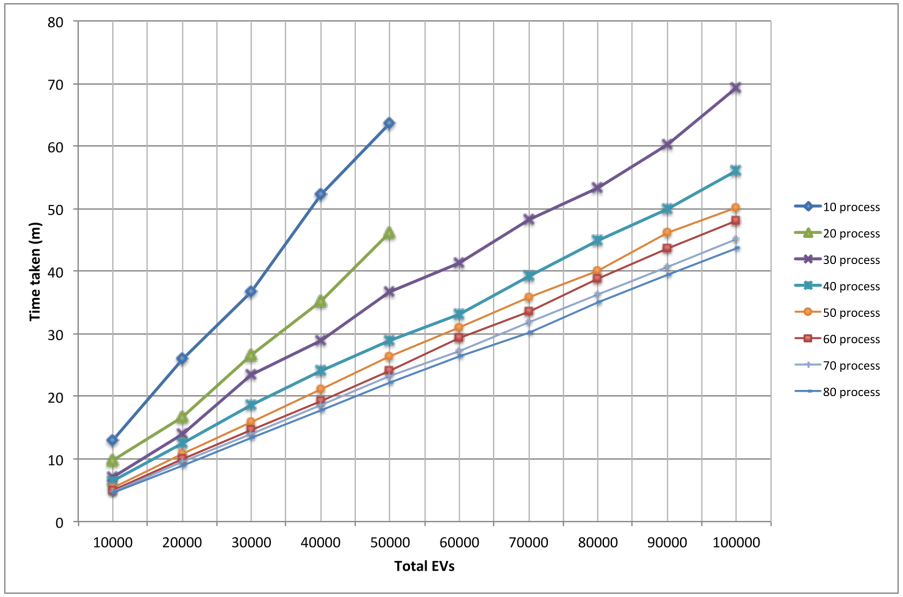
Master Procesing Time (m)
25
24
23
22
21
20
19
18
17
16
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
0
10000
20000
30000
40000
50000
60000
70000
Total EVs
10 process
20 process
30 process



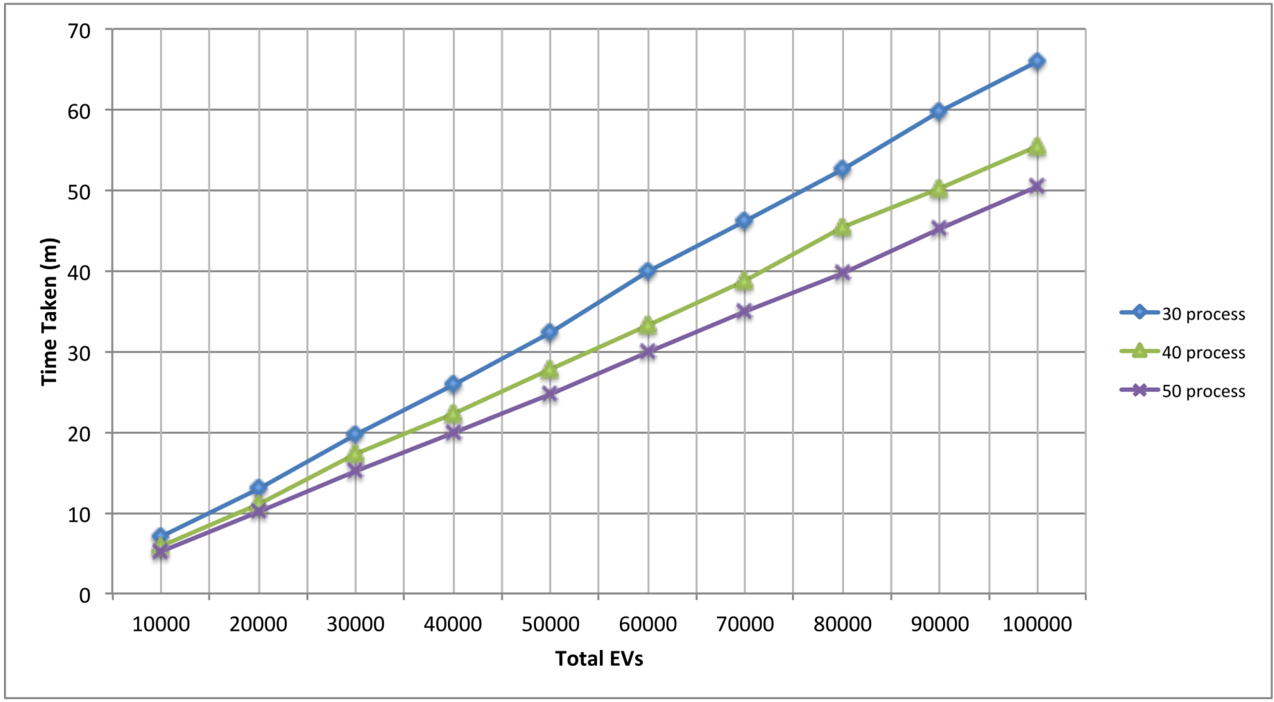
Time taken (m)
20
18
16
14
12
10
8
6
4
2
0
10000
20000
30000
40000
50000
60000
70000
80000
90000
100000
Total EVs
10 process
20 process
30 process
40 process
50 process
60 process
70 process
80 process



45
40
35
30
25
20
15
10
5
0
Master Processing Time (s)
10000
20000
30000
40000
50000
60000
70000
80000
90000
100000
Total EVs
30 process
40 process
50 process



80
70
60
50
40
30
20
10
0
Time taken (m)
10000
20000
30000
40000
50000
60000
70000
80000
90000
100000
Total EVs
10 process
20 process
30 process
40 process
50 process
60 process
70 process
80 process



70
60
50
40
30
20
10
0
Time Taken (m)
10000
20000
30000
40000
50000
60000
70000
80000
90000
100000
Total EVs
30 process
40 process
50 process

Algorithm

procedure VE($max\_iter, total\_ev, \rho, input\_data\_ev, input\_data\_aggregator,$

$u, \bar{x}, x^* \leftarrow 0$

for each k in max_iter **do**

for each EV i in $total\_ev$ **do**

$data_i \leftarrow$ Read i-th input from $input\_data\_ev$

$x^* \leftarrow$ PerformEVOptimization($x^*, u, \bar{x}, data_i$)

$X_{EV}[\mathrm{i}] \leftarrow$ Store x^* against EV i

$sum_{ev} \leftarrow$ Calculate sum of EV profiles $\sum X_{EV}$

$data_{agg} \leftarrow$ Read aggregator data from $input\_data\_aggregator$

$x^*_{agg} \leftarrow$ PerformAggregatorOptimization($x^*_{agg}, u, \bar{x}, data_{agg}$)

$\bar{x} \leftarrow (sum_{ev} + x^*_{agg})/total\_ev$

$u \leftarrow u + \bar{x}$

Calculate cost

if Converged() **then**

break

Algorithm

procedure VE(max_iter,$total\_ev$,ρ,$input\_data\_ev$,$input\_data\_aggregator$,
    $u, \overline{x}, x^* \leftarrow 0$
    for each k in max_iter **do**
        for each EV i in $total\_ev$ **do**
            $data_i \leftarrow$ Read i-th input from $input\_data\_ev$
            $x^* \leftarrow$ PerformEVOptimization($x^*, u, \overline{x}, data_i$)
            $X_{EV}[\mathrm{i}] \leftarrow$ Store x^* against EV i
        $sum_{ev} \leftarrow$ Calculate sum of EV profiles $\sum X_{EV}$
        $data_{agg} \leftarrow$ Read aggregator data from $input\_data\_aggregator$
        $x^*_{agg} \leftarrow$ PerformAggregatorOptimization($x^*_{agg}, u, \overline{x}, data_{agg}$)
        $\overline{x} \leftarrow (sum_{ev} + x^*_{agg})/total\_ev$
        $u \leftarrow u + \overline{x}$
        Calculate cost
        if Converged() **then**
            break

Algorithm

procedure VE(*max_iter*,*total_ev*,ρ,*input_data_ev*,*input_data_aggregator*,
    $u, \overline{x}, x^* \leftarrow 0$
    for each k in *max_iter* **do**
        for each EV i in *total_ev* **do**
            $data_i \leftarrow$ Read i-th input from *input_data_ev*
            $x^* \leftarrow$ PerformEVOptimization(x^*, u, $\overline{x}$, $data_i$)
            X_{EV}[i] $\leftarrow$ Store x^* against EV i
        $sum_{ev} \leftarrow$ Calculate sum of EV profiles $\sum X_{EV}$
        $data_{agg} \leftarrow$ Read aggregator data from *input_data_aggregator*
        $x^*_{agg} \leftarrow$ PerformAggregatorOptimization(x^*_{agg}, u, $\overline{x}$, $data_{agg}$)
        $\overline{x} \leftarrow (sum_{ev} + x^*_{agg})/total\_ev$
        $u \leftarrow u + \overline{x}$
        Calculate cost
        if Converged() **then**
            break

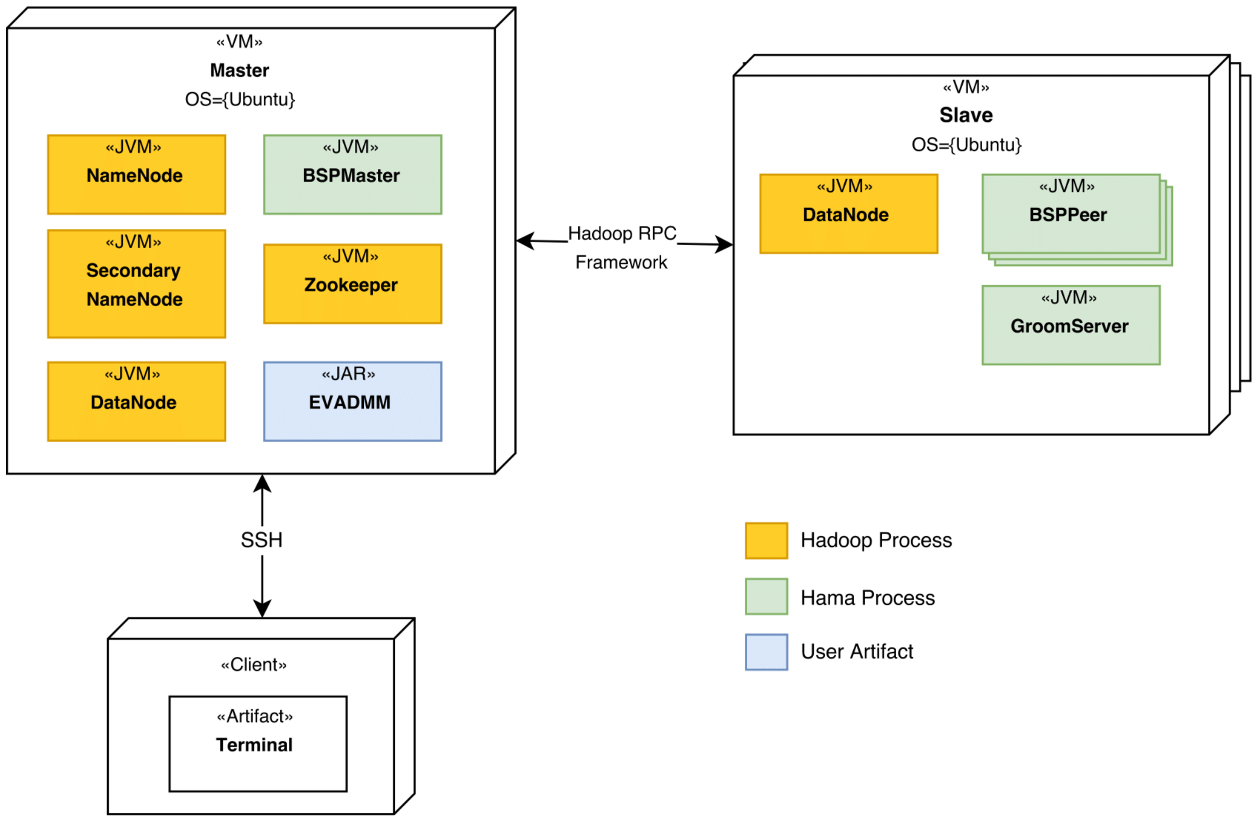
Runs in ***Parallel*** on ***Slaves***

LRZ

- ▷ Leibnitz Rechen Zentrum
- ▷ Infrastructure as a Service
- ▷ flexible, secure, highly available

- ▷ All Docker problems gone !

Deployment


«VM»
Master
OS={Ubuntu}
«JVM»
NameNode
«JVM»
BSPMaster
«JVM»
Secondary NameNode
«JVM»
Zookeeper
«JVM»
DataNode
«JAR»
EVADMM
Hadoop RPC Framework
«VM»
Slave
OS={Ubuntu}
«JVM»
DataNode
«JVM»
BSPPeer
«JVM»
GroomServer
SSH
«Client»
«Artifact»
Terminal
Hadoop Process
Hama Process
User Artifact

Algorithm

procedure VE($max\_iter, total\_ev, \rho, input\_data\_ev, input\_data\_aggregator,$

  $u, \bar{x}, x^* \leftarrow 0$

  for each k in max_iter **do**

    for each EV i in $total\_ev$ **do**

      $data_i \leftarrow$ Read i-th input from $input\_data\_ev$

      $x^* \leftarrow$ PerformEVOptimization($x^*, u, \bar{x}, data_i$)

      $X_{EV}[\mathrm{i}] \leftarrow$ Store x^* against EV i

    $sum_{ev} \leftarrow$ Calculate sum of EV profiles $\sum X_{EV}$

    $data_{agg} \leftarrow$ Read aggregator data from $input\_data\_aggregator$

    $x^*_{agg} \leftarrow$ PerformAggregatorOptimization($x^*_{agg}, u, \bar{x}, data_{agg}$)

    $\bar{x} \leftarrow (sum_{ev} + x^*_{agg})/total\_ev$

    $u \leftarrow u + \bar{x}$

    Calculate cost

    if Converged() **then**

      break

Runs in ***Parallel*** on ***Slaves***

Algorithm

Abstraction on distributed system

```java
ADFJob job = new ADFJob();

job.setMaxIteration(4);
job.setJobName("ADF Exchage EVADMM job");
job.setInputPath("aggregator.txt,EVs.txt");
job.setOutputPath("output/");
job.setSolutionVectorSize(96);

job.setADMMClass(BSPExchange.class);
job.setFunction1(ValleyFillingOptimizationFunction.class);
job.setFunction2(EVOptimizationFunction.class);

job.run();
```

Disclaimer: This is an alpha version.