LEMO-MR: Low overhead and Elastic MapReduce Implementation Optimized for Memory and CPU-Intensive Applications

Zacharia Fadika, Madhusudhan Govindaraju

Department of Computer Science
State University of New York (SUNY) at Binghamton

MAPREDUCE

- Programming model
  - For processing large data set
  - Exploits a large set of commodity computers
  - Implemented in 2004 in the form of Hadoop
- Characteristics of the model
  - Relaxed synchronization constraints
  - Locality optimization
  - Fault-tolerance
  - Load balancing
- Map/Reduce
  - Inspired from functional programming
  - The map() function is called on every item in the input and emits a series of intermediate key/value pairs
  - All values associated with a given key are grouped together
  - The reduce() function is called on every unique key, and its value list, and emits a value that is added to the output
- More formally
  - Map(k1,v1) --> list(k2,v2)
  - Reduce(k2, list(v2)) --> list(v2)

CLUSTER ORGANIZATION

- Master/Worker
  - 1 Master
    - Handles cluster I/O
    - Track workers
    - Tracks job's health
    - Scedules job re-executions
  - Multitude of workers
    - Keep in touch with the master
    - Work independently of each other

HADOOP MAPREDUCE

- Introduced in 2004 by Dean and Ghemawat
- Suited for Data Intensive applications
- Relies on the HDFS for:
  - Input split management
  - Load-balancing
  - Fault-tolerance
- DataNodes host HDFS blocks
  - Send updates to the Master about block conditions
  - Necessary feature for fault-tolerance, provides locality
  - Costly feature nonetheless
- Tasktrackers accept tasks
  - Send continuous updates to the Master about task progress
  - Costly feature with a high number of mappers

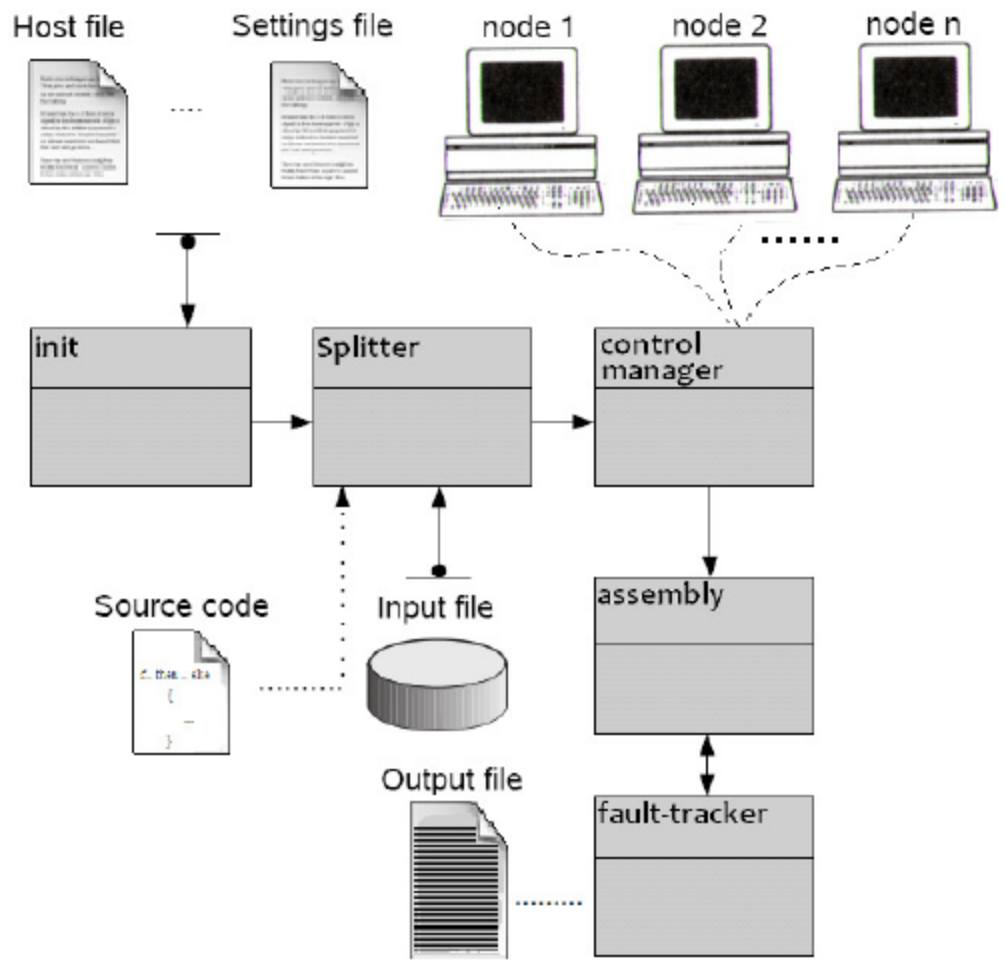
MAPREDUCE AND CPU-INTENSIVE APPLICATIONS

- CPU-intensive applications
  - Predominant in scientific applications.
  - Require more processing per item than average MapReduce applications (Url count, triages)
  - 9400 nanosecond vs 1 millisecond
- LEMO-MR
  - Low overhead nature is suited for CPU-intensive applications
  - Built and benchmarked against a “perfect” MapReduce system with no networking overhead.

LEMO-MR

- Low overhead
  - Limited node to node communication
  - No data shuffling
  - Passive fault-tolerance
- Elastic
  - Statically elastic
    - Enabled by input and node decoupling
    - Expansion occurs seamlessly in between jobs without rebalancing the cluster
- For in-Memory and CPU-intensive applications
  - LEMO-MR deals directly with the native filesystem
  - Input is abstracted to allow diskless systems to be used through input streaming.

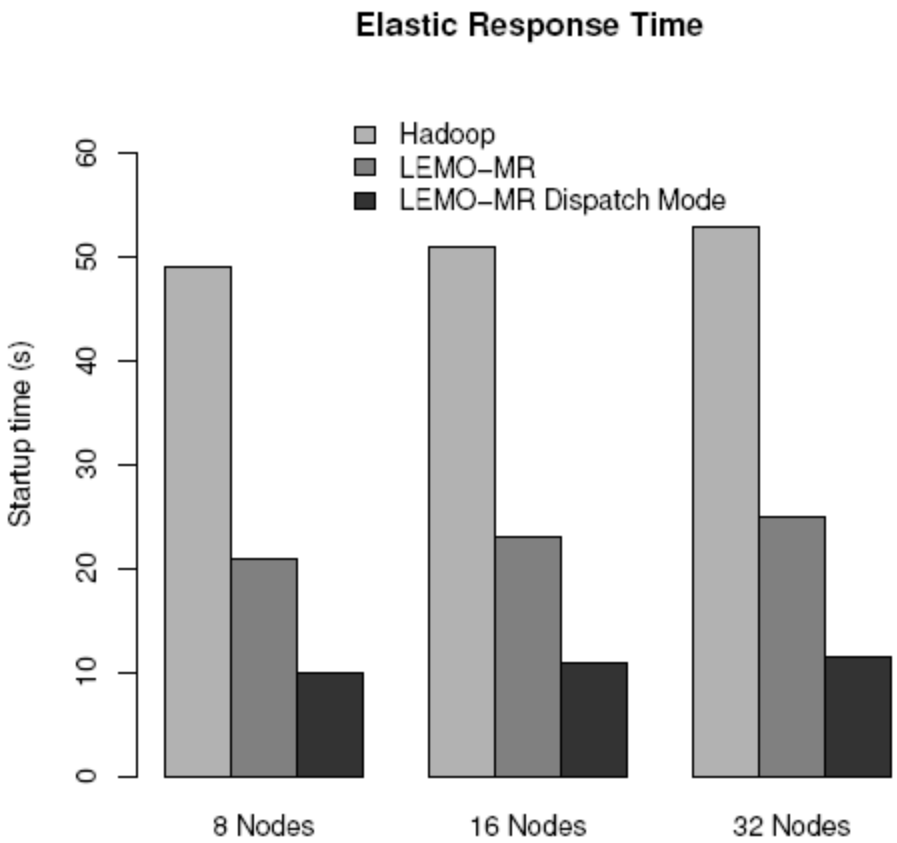
LEMO-MR ARCHITECTURE



LEMO-MR ARCHITECTURE

- Init module
  - Gathers information from the master node
    - Node list, number of nodes to run, Input location … etc
- Splitter module
  - Acts as a file manager or as a mere splitter
  - Splits are controlled from the master and not replicated through the cluster nodes
- Control Manager
  - Launches and monitors errors
- Fault-Tracker
  - Listens for errors and failure messages
- Assembly
  - Applies the reduce function and reassembles the output

LEMO-MR ELASTIC PROPERTY



- Input decoupling from nodes allows for system to be grown or shrunk with minimal delay.
- No Input rebalancing necessary

TEST ENVIRONMENT

- We run our tests on a selection of two machine classes:
  - 1× dual core – One desktop-class machine, which has a single 2.4Ghz Intel Core2 6600 with 2GB of ECC RAM,running Linux 2.6.24. The filesystem in use here is ext3fs.
  - 2× uniprocessor – 1U nodes in a cluster, each of which has two 3.2Ghz Intel Xeon CPUs, 4 gigabytes of RAM and run a 64 bit version of Linux 2.6.15. The filesystem in use in the test directory here is reiserfs.

EXPERIMENTAL SETUP

- The number of workers used throughout the experiments varied from a single node to 64 nodes.
- The Master does not share any of the work performed by the workers. Its sole role is that of supervision, input distribution, and output collection.
- A battery of configurable, automated scripts on the Master triggers the experiments and records the results
- These scripts serve as a lab book, and can be consulted for replication of experiments as well as documentation of each experiment.

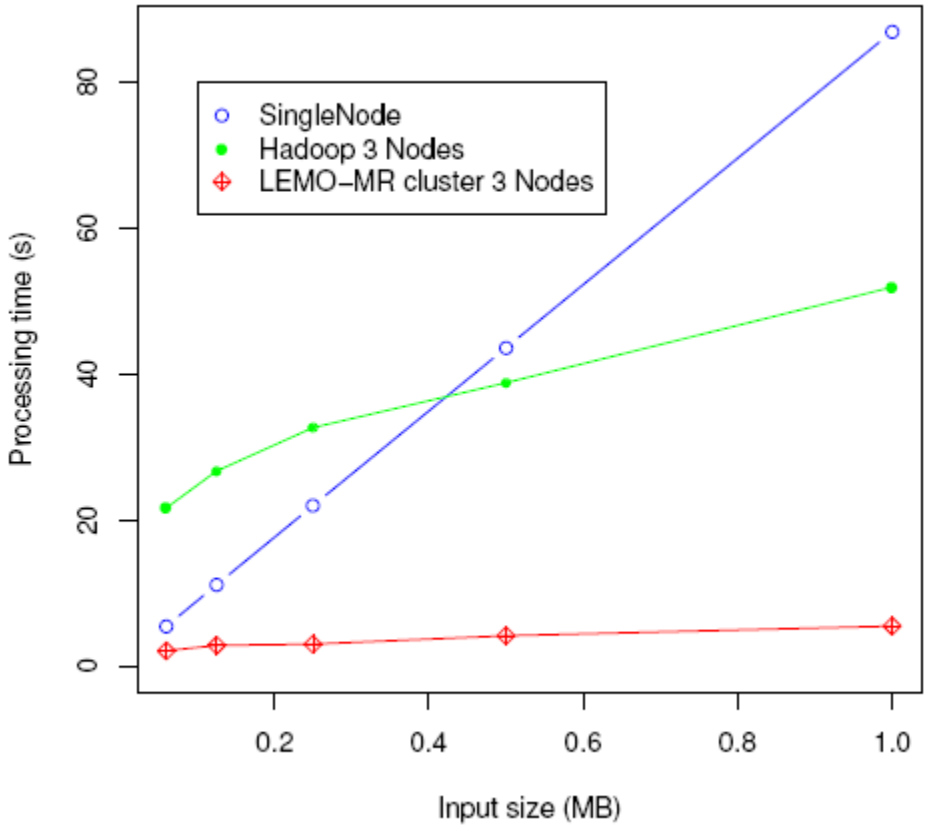
EXPERIMENTAL SETUP

- Both frameworks use identical nodes, identical input datasets and similar source code
- Processing is that of arrays of doubles encoded in XML files
- For XML parsing, AxisJava was used

AxisJava is a SOAP engine which allows for the creation of SOAP processors used in the parsing of XML datasets.

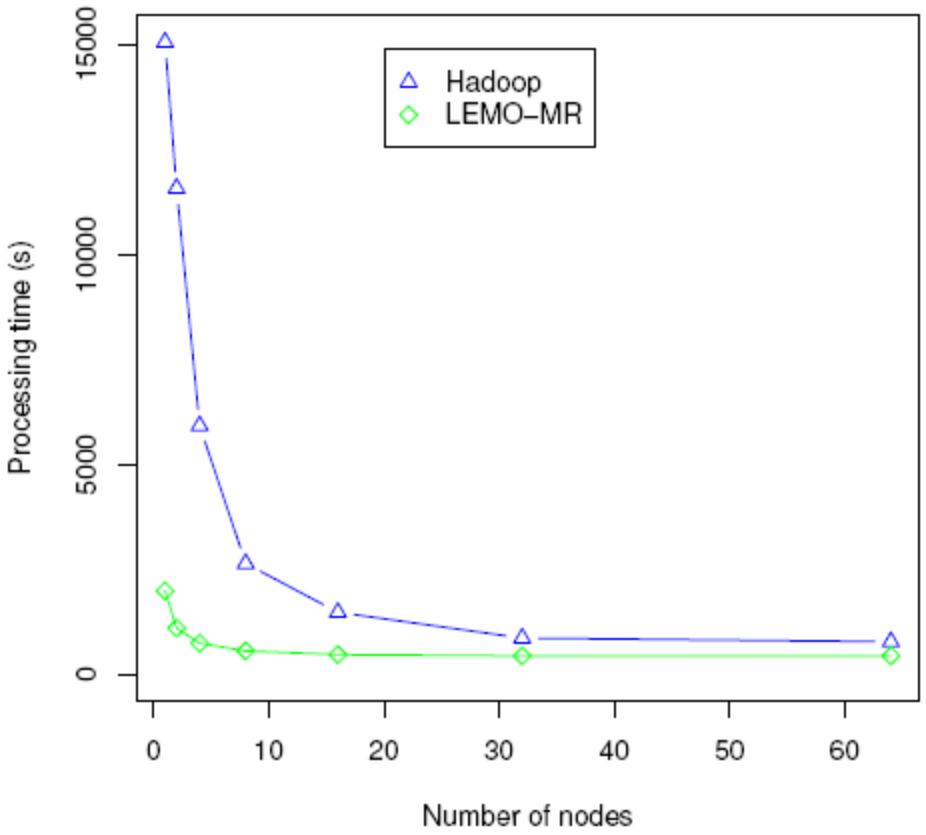
AxisJava's SOAP processor was extracted from the toolkit, and its code, ported to the Hadoop and LEMO-MR applications developed

SINGLE NODE PROCESSING VS 3 NODE CLUSTERS



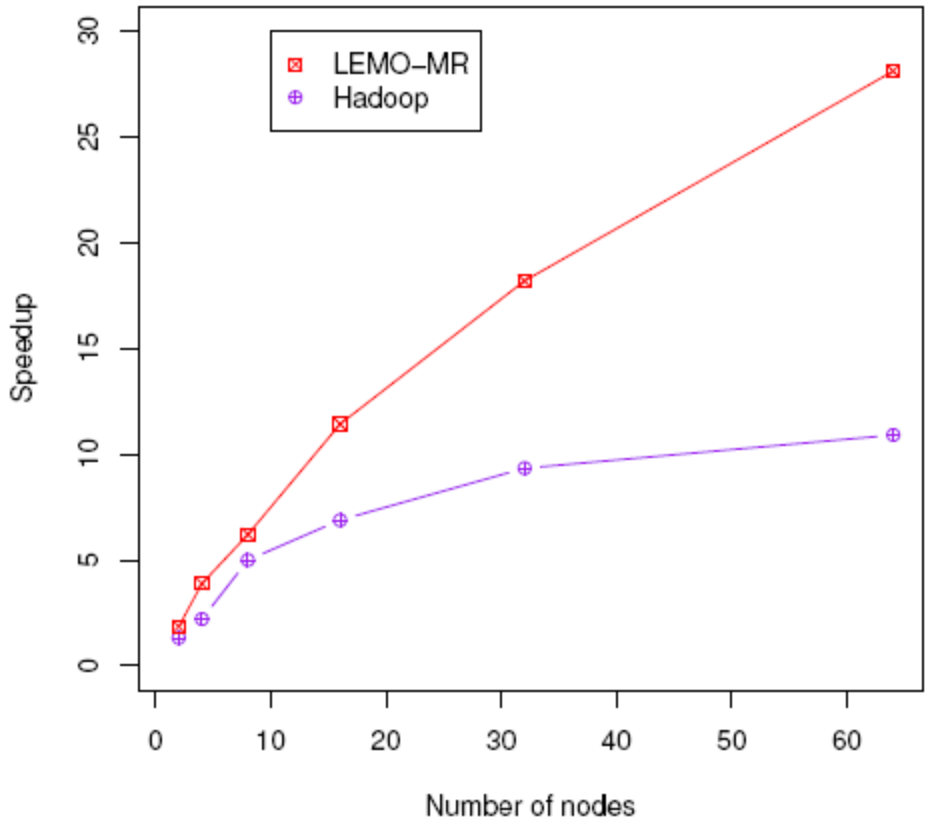
- **Single node processing versus 3-node cluster**
- **The cluster overpowers the single node system around 500 KB of data**
- **Latencies and overhead are offset as the data size grows**

LEMO-MR IS 1.5 TIMES FASTER WITH 64 NODES



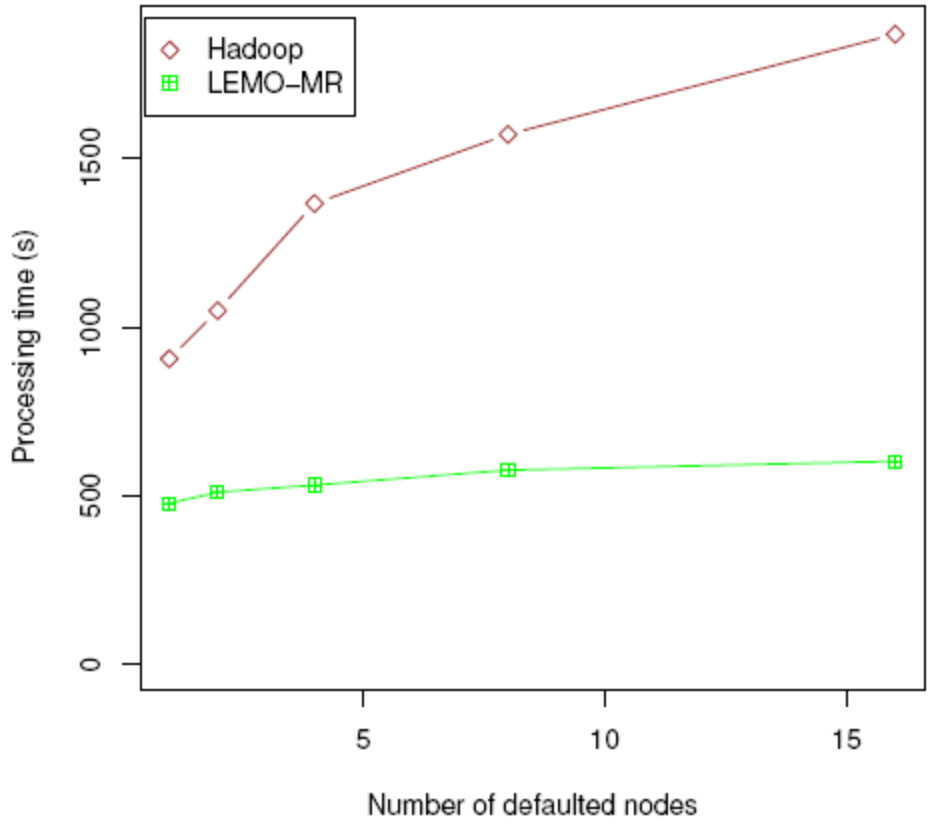
- **Nodes are increased with both clusters for 2.5GB of data**
- **For a low number of nodes, the difference between both frameworks is significant**
- **As more nodes come in, both frameworks regain some stability**
- **This shows that to offset MapReduce overhead, either more work is required, or more nodes are required.**
- **LEMO-MR however shows more stability.**
- **After a certain point, cluster optimality is reached. This exists for every MapReduce job**
- **LEMO-MR is 1.5 times faster with 64 Nodes**

LEMO-MR SCALES BETTER THAN HADOOP



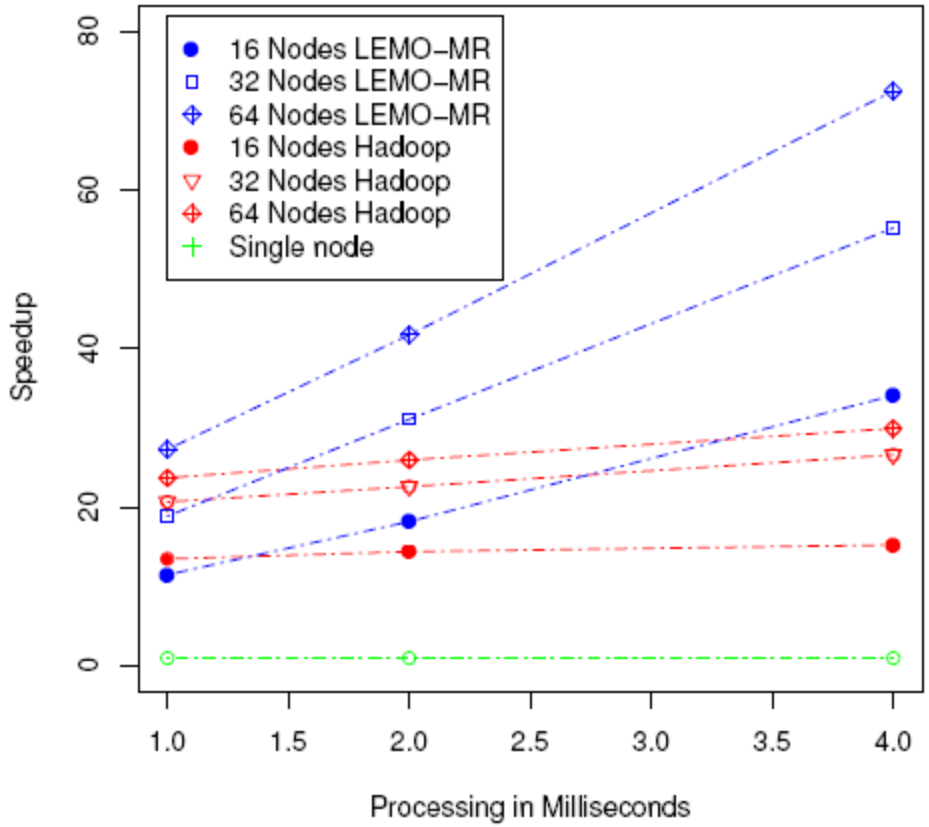
- **2.5GB, 240 Million elements processed**
- **Nodes are gradually added to both clusters, and speed up is measured as *TsingleNode / TMapReduce*. The single node setups represent each cluster with only a single member.**
- **This shows the scalability of both frameworks with respect to each other.**

FAULT-TOLERANCE AS NODES DEFAULT



- **Application runtime in the face of node failures. 2, 4, 8 and 16 nodes are lead to fail.**
- **This graph is a test of our Fault-tolerance mechanism**
- **Cluster size is 60 nodes. Data size is 2.5GB**
- **LEMO-MR performance is explained by the fact that it takes the same work rescheduling failed work as it does scheduling newly assigned work.**

MAPREDUCE FOR CPU INTENSIVE JOBS



- **Processing intensity increase graph**
- **Both frameworks are tested with different work intensities in diverse node configurations against their single node counterparts.**
- **This tests how both scale with more processing per item.**
- **Hadoop briefly does better with 16 and 32 node clusters, not so with the 64 node cluster because LEMO-MR's performance against its single node counterpart is constant. Hadoop scales up dramatically with less intensity per item.**

CONCLUSIONS

- For an application to benefit from the model, processing done by such application must trump the amount of overhead, produced by it.
- The MapReduce model is not only suited to Data Intensive applications, but also for CPU-intensive cases.
- Sheer data size is not the only factor affecting MR performance.
- A low-overhead approach to MapReduce can prove itself beneficial for CPU-intensive applications, more so than Data intensive applications as latencies are accentuated in CPU-intensive applications.
- LEMO-MR allows compute node independence and decoupling, allows cluster elasticity at very low cost.
- Reducing node to node communication can still allow for an effective framework while minimizing performance dampening factors
- Fault-tolerance can be achieved in a passive manner while still maintaining effectiveness.

FUTURE PLANS

- WE PLAN TO RELEASE LEMO-MR SOON