VNetOS: Virtualised Distributed and Parallel Sensor Network Operating Environment for the IoT and SHM

For Educatuion and Research

Stefan Bosse [1,*]

[1]University of Bremen, Dept. Mathematics & Computer Science, Bremen, Germany

[*]Presenting author

Overview

How can we compose and evaluate distributed sensor networks?

Overview

How can we compose and evaluate distributed sensor networks?

How can we design an universal and simple operating system for distributed sensor networks?

Overview

How can we compose and evaluate distributed sensor networks?

How can we design an universal and simple operating system for distributed sensor networks?

We have to address education as well as research as well as application!

Challenges

- Dealing with **distributed and parallel computing in strong heterogeneous environments**, e.g., distributed sensor networks, is still a challenge at the:
    - algorithmic,
    - communication, and
    - application levels.
- **Heterogeneity** is related to different computer and network (communication) architectures
- **Virtualization** can hide and unify heterogeneity.

Challenges

Besides inter-process communication and synchronization, the unified access and monitoring of computing nodes (devices, computers, processors) is required to handle distributed and parallel systems in a comfortable and easy-to-access manner.

Summary and Highlights

In this work, a unified distributed and parallel framework and Web tools are introduced using **Virtual Machines** (VM) and **Web browsers** to control them.

1. The framework enables the control, monitoring, and study of distributed-parallel systems, especially addressing sensor networks and IoT networks.

2. Nodes can be arranged in a graphical drawing world or script-based.

Summary and Highlights

3. Virtual network nodes are assigned to VM instances that can be created inside the browser using WebWorker processes or can be attached to externally running VM instances via a Web control API.

4. New VM instances or processes can be started and controlled instantly.

5. The graphical UI provides access to the internal and external nodes, programming editors, and monitor shells.

6. The VMs can be generic, but in this work there is a focus on JavaScript and Lua.

    - The framework provides augmented virtuality, i.e., a coupling of physical and virtual worlds

Network Architecture

The general architecture consists of a generic network graph $G=\langle N,P,C\rangle$ with VM nodes N that can process a textual programming language L, a set of communication ports P attached to nodes, and communication connections (links) L between ports.

Hardware and Software - Complete

Three different programming language VMs are considered in this work:

1. JavaScript (using V8/Spidermonkey/jerryscript [JERR] engines);
2. Lua (C-Lua, eLua, LuaJit [LUAJ], Fengari Web/VM-in-VM [FENG]);
3. JavaScript Agent Machine (JAM, programmed in JS, VM-in-VM, [BOS18A]).

Three different host computers are considered:

1. Generic desktop and mobile laptop computers (x86,x64, 2 cores, 2GHz clock) supporting all VMs;
2. Embedded system Raspberry PI (Zero, 3, Arm, 1-2 cores, 1GHz clock) supporting all VMs;
3. Tiny embedded system ESP32 (Tensilica, 2 cores, 240MHz clock) supporting Lua primarily and JS VMs secondarily.

Hardware and Software - Complete

There are two meta classes of VM APIs used in the framework:

A. A root meta VM that is the main process providing a Web RPC API to create and control worker processes;
B. The real target VM (JS, Lua, FORTH, ...) that is executed in a worker process, providing an RPC service (especially for IO), too.

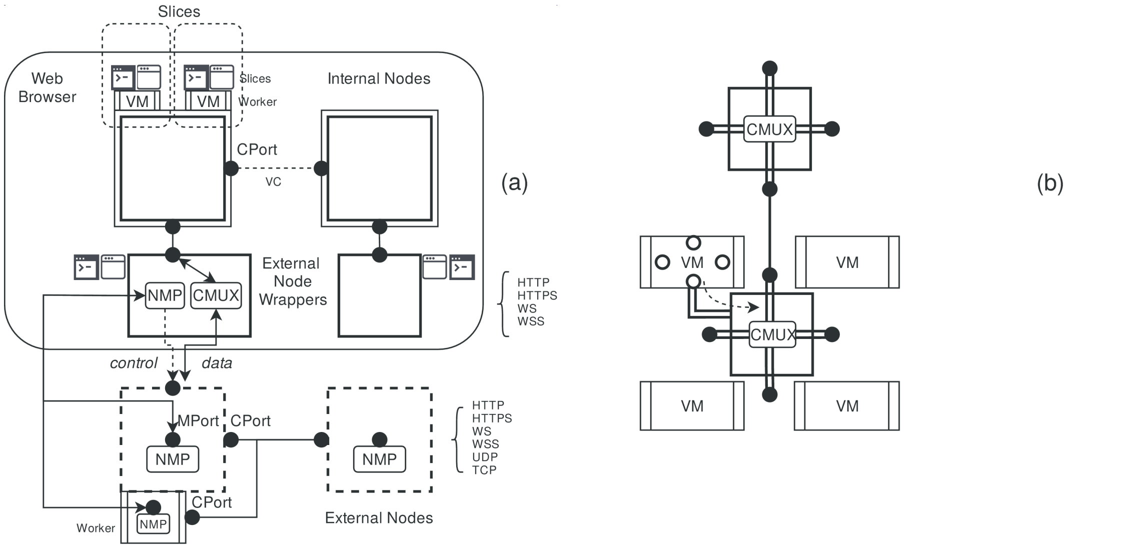


Fig. 1. (a) General software framework and communication architecture with internal (double outline), external mapper (single outline), and external (dashed line) nodes. There are management communication ports (mPort) for connecting Web controllers with external nodes and generic communication ports (cPort) for inter-node communication (b) Message multiplexer architecture

Software Framework

VNetOS

The VNetOS software framework consists of the following parts:

1. The Web browser GUI application with a 2D graphical network world consisting of
    - graphical node entities with communication links between nodes,
    - code editors with syntax highlighting,
    - process monitor and interactive shell windows,
    - and external node controllers;
2. Internal VMs that can be embedded in the Web browser, i.e., can be provided in JavaScript or WebAssembly;
3. External VMs with a Web RPC service that have virtual shadow nodes in the Web GUI;
4. A set of programming modules supporting parallel and distributed programming (like CSP modelling, sensor access, RPC; for each target VM language there is an implementation).

Nodes

Three different node classes are distinguished:

I-Node
Internal node with an embedded meta VM processed by the browser JS VM.

P-Node
External node processed by a native VM.

V-Node
Virtual wrapper (twin) of a p-node in the Web browser with NMP access and control.

OS JS

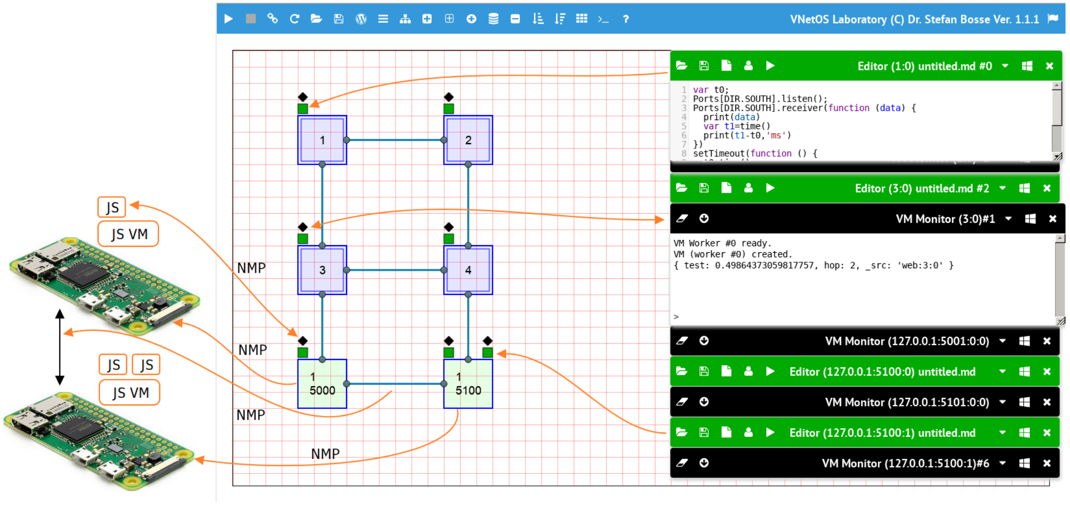
Preliminary Experiments and Results

Experiments

Three principle experiments were performed:

1. Network of 16/8 internal nodes arranged in a 2D mesh-grid;
2. Network of 16/8 external nodes arranged in a 2D mesh-grid;
3. Hybrid 8 internal + 8 external nodes.

Example


VNetOS Laboratory (C) Dr. Stefan Bosse Ver. 1.1.1
Editor (1:0) untitled.md #0
Editor (3:0) untitled.md #2
VM Monitor (3:0)#1
VM Monitor (127.0.0.1:5001:0:0)
Editor (127.0.0.1:5100:0) untitled.md
VM Monitor (127.0.0.1:5101:0:0)
Editor (127.0.0.1:5100:1) untitled.md
VM Monitor (127.0.0.1:5100:1)#6
1
2
3
4
1
5000
1
5100
NMP
NMP
NMP
NMP
JS
JS VM
JS
JS
JS VM

```
var t0;
Ports[DIR.SOUTH].listen();
Ports[DIR.SOUTH].receiver(function (data) {
  print(data)
  var t1=time()
  print(t1-t0,'ms')
})
setTimeout(function () {
```

```
VM Worker #0 ready.
VM (worker #0) created.
{ test: 0.49864373059817757, hop: 2, _src: 'web:3:0' }
>
```

Fig. 2. Typical network application using VNetOS, a Web browser, and Raspberry PI Zero devices: Four internal and two external nodes connected via WLAN. For each VM instance there is a code editor and an IO monitor shell window. Internal and external nodes can communicate directly via HTTP.

Results

Host	dhry/s	VM	t_{iVM}	m_{iVM}	t_{cmsg}
PC/nodejs	5000k	JS (ext)	140ms	20MB	3ms
PC/Firefox	4200k	JS (int)	100ms	10MB	4ms
PC/plvm	600k	Lua, Parallel LuaJit(+libuv) (ext)	3ms	800kB	0.1ms
Raspberry PI Zero/nodejs	230k	JS	1600ms	20MB	40ms
Raspberry PI Zero/plvm	40k	Lua, Parallel LuaJit(+libuv) (ext)	10ms	800kB	1ms
ESP32/Lua	1k	Lua, FreeRTOS (ext)	100ms	100kB	5ms

Tab. 1. m_{iVM}: Base memory (RAM+ROM), t_{iVM}: VM Instantiation time, t_{cmsg}:Communication time between nodes

Results

1. Lua can be easily embedded and forked using multi-threading, whereas node.js requires system process creation (at least some time ago), resulting in an instance creation time 100 times higher.

2. Communication time is limited due to core bandwidth/latency and by the process/thread scheduling times required for message multiplexer invocation.

3. Lua (LuaJit) shows superior performance compared to node.js/V8-based VMs and is a suitable VM for (tiny) embedded systems.

4. The base memory requirement for node.js (and Web browser engines) pose the highest start-up times and memory requirements, but also the highest computational power.

Conclusions

- A novel distributed virtualization framework for the deployment and control of heterogeneous networks of generic and embedded systems was introduced.
- The control of the distributed network is performed by a graphical Web browser application (or alternatively, script-based).
  - Via the Web application, each node can be controlled by the NMP protocol.
  - Each physical node has a virtual representation in the Web application
  - The physical and virtual nodes are connected via NMP.

- Each root node supports a programmable target VM (e.g., JS, Lua) and can instantiate (fork) VM worker processes.
    - VM instances can be connected with each other by using generic communication ports.
- The routing of messages is performed by a message router.
- Evaluation of the node performance identified VM forking and message routing times as critical, but strongly dependent on the underlying VM (LuaJit forking is 100 times faster than node.js).

Even tiny embedded systems can be used for distributed programming and processing. Besides education, simulation and generic distributed network control are core applications.

VNetOS: Virtualised Distributed and Parallel Sensor Network Operating Environment for the IoT and SHM

For Educatuion and Research

Stefan Bosse [1,*]

[1]University of Bremen, Dept. Mathematics & Computer Science, Bremen, Germany

[*]Presenting author