DynaMix: Dynamic Mobile Device Integration for Efficient Cross-device Resource Sharing

Dongju Chae\textsuperscript{1}, Joonsung Kim\textsuperscript{2}, Gwangmu Lee\textsuperscript{2}, Hanjun Kim\textsuperscript{1}, Kyung-Ah Chang\textsuperscript{3}, Hyogun Lee\textsuperscript{3}, and Jangwoo Kim\textsuperscript{2}

\textsuperscript{1}Dept. of Computer Science and Engineering, POSTECH
\textsuperscript{2}Dept. of Electrical and Computer Engineering, Seoul National University
\textsuperscript{3}Samsung Electronics
Increasing Number of IoT Devices
Sharing Multi-device Enables Many Services

I want to build my home theater

Audio

Custom Home Theater

Screen

Storage

Audio

Smart Monitoring System?

Remote Gaming System?

..., and various custom services.
Index

• **Limitations of existing schemes**
  – I/O request forwarding
  – Manual programming

• **DynaMix: Efficient dynamic resource sharing**

• **Evaluation**

• **Conclusion**
Running Example: Home Theater

open(storage);
load(data);
decode(data);
render(data);

Program
Loader
Decoder
(1) I/O Request Forwarding
- with the example of **home theater**

```
open(storage);
load(data);
decode(data);
render(data);
```

---

**Diagram:**

- Phone
  - Loader
  - Decoder
  - Program

- TV
  - Storage
  - Screen

**I/O Resources**

- Data
  - High Traffic!
(1) I/O Request Forwarding
- with the example of **home theater**

**Good**

I/O abstraction layer for transparency

✓ **Easy programming environment**

**Bad**

High network traffic due to unoptimized datapath

✓ **Poor performance**
(2) Manual Programming
- with the example of **home theater**
(2) Manual Programming
- with the example of **home theater**

### Good

Device-aware task partitioning
- **Good performance**

### Bad

Hand-tuned multi-device application
- **High programming effort**
Design Goals

- Programmability
- Performance

- Request Forwarding
- DynaMix
  High Performance & Programmability
- Manual Programming
Index

- Limitations of existing schemes

- **DynaMix: Efficient dynamic resource sharing**
  - Key ideas
  - Architecture & Implementation

- Evaluation

- Conclusion
Key Ideas of *DynaMix*

1) Transparent & Wide Resource Integration
   - Kernel-level resource integration
     - Transparent integration (easy programmability)
   - Sharing multiple types of resources (i.e., CPU, Memory, I/O)
     - Wide resource coverage

2) Resource-aware Dynamic Task Redistribution
   - Contention detection based on resource usage
   - Performance estimation for migration scenarios
     - Optimized performance
Key Ideas of *DynaMix* (1/2)

1) Transparent & Wide Resource Integration

I want to use remote screen on TV!
Key Ideas of DynaMix (2/2)

2) Resource-aware Dynamic Task Redistribution

![Diagram showing resource-aware task redistribution](image-url)
Key Ideas of *DynaMix* (2/2)

2) Resource-aware Dynamic Task Redistribution

![Diagram](image)

- **Loader**
  - **DynaMix**
  - **I/O Resource**
  - **Phone**
  - **Low Traffic!**
  - **Decoder**
    - **DynaMix**
    - **I/O Resource**
    - **TV**

**I/O datapath**
- Local I/O
- Remote I/O

**User Kernel**
DynaMix Architecture

(1)
DynaMix Architecture

Loader  Decoder

Migration Selector

Contention Detector

Resource Integrator

Storage  Screen

Local Device (Master)

Screen

Remote Device

User
Kernel

(2)

DynaMix

DynaMix

Migration Selector

Contention Detector

Resource Integrator

(2)
DynaMix Architecture

Loader

Decoder

User

Kernel

DynaMix

Migration Selector

Contention Detector

Resource Integrator

Storage

Screen

Local Device (Master)

Remote Device

Screen

Kernel

18/35
(1) Resource Integrator – (Memory, CPU, I/O)

- **Memory integration**
  - Distributed shared memory (DSM)

- **Three perf. optimizations**
  - Lazy release consistency model
  - Page-level coherency block
  - Memory prefetching
(1) Resource Integrator – (Memory, **CPU**, I/O)

- **CPU integration**
  - Thread migration

- **Optimizations**
  - Thread group granularity
  - Clone-based migration
  - Transparent live migration
(1) Resource Integrator – (Memory, CPU, I/O)

- I/O resource integration
  - Request forwarding (e.g., device file)

- Optimizations
  - Data compression
  - Platform-assisted handling
2) Contention Detector

- Calculate the slowdown of each thread
  - Collect per-thread resource usage (e.g., CPU, network, ...)
  - Measure a stall time due to resource access
2) Contention Detector

- Calculate the slowdown of each thread
  - Collect per-thread resource usage (e.g., CPU, network, ...)
  - Measure a stall time due to resource access
3) Migration Selector

- **Estimate the performance tradeoff after migration**
  - Utilize *intra-/inter-device information* to decide migration targets

1. Intra-device communication
2. Inter-device bandwidth & latency
3. Remote resource status

- **Diagram:**
  - Group 1: Loader 2MB, Storage 2MB
  - Group 2: Decoder 5MB, Player 5MB, Screen 5MB
  - Phone: TV
3) Migration Selector

- **Estimate the performance tradeoff after migration**
  - Utilize intra-/inter-device information to decide migration targets
  - Calculate the tradeoffs of all possible migration scenarios
Index

• Limitations of existing schemes
• DynaMix: Efficient dynamic resource sharing

• Evaluation
  - Scenario #1: Home theater
  - Scenario #2: Smart monitoring

• Conclusion
Evaluation Setup: Home theater

Home Theater

- Large TV (Remote)
- HQ Speaker (Remote)
- Master Device (Local)
- Android-based smartphones (5.1.1)
  - Nexus 5 (master)
  - Nexus 4 (connected to a speaker)
- Tizen-based smart TV (2.3)

Phone (Master)

- Loader
- Decoder
- Storage
- Screen
- Audio

Speaker

Audio

Screen

TV
Performance Timeline

- Resource reconfiguration (local screen → remote large screen)

Local Screen
- Select a remote large screen
- Detect contention & Start analysis

Remote Large Screen
- Start migration
- Restore the target performance

Frames per second (FPS)

Time (sec)
Throughput Improvement
- Measured FPS for the home theater scenario

• **Achieve (or closely) the target performance goal**

![Graph showing Throughput Improvement](image)

- Ideal (24 FPS)
- Request Forwarding
- DynaMix

![Video quality and Frames Per Second](image)

- 240p
- 360p
- 480p
- 720p (HD)
- 1080p (Full HD)

- **Frames Per Second**
  - 30
  - 24
  - 18
  - 12
  - 6
  - 0

- **Video quality**
  - 240p
  - 360p
  - 480p
  - 720p (HD)
  - 1080p (Full HD)

- **Request Forwarding**
  - 4x higher

- **DynaMix**
  - 8.3x higher
Minimized Network Stall Time
- Per-frame latency analysis for each datapath

- **Reduce the exposed network traffic**

---

**DynaMix** (DM)

**Request Forwarding** (RF)

---

<table>
<thead>
<tr>
<th>Video quality</th>
<th>RF</th>
<th>DM</th>
<th>RF</th>
<th>DM</th>
<th>RF</th>
<th>DM</th>
</tr>
</thead>
<tbody>
<tr>
<td>240p</td>
<td>30</td>
<td>30</td>
<td>30</td>
<td>35</td>
<td>30</td>
<td>35</td>
</tr>
<tr>
<td>480p</td>
<td>48</td>
<td>48</td>
<td>48</td>
<td>48</td>
<td>48</td>
<td>48</td>
</tr>
<tr>
<td>720p (HD)</td>
<td>347</td>
<td>347</td>
<td>347</td>
<td>347</td>
<td>347</td>
<td>347</td>
</tr>
<tr>
<td>1080p (Full HD)</td>
<td>5.3</td>
<td>5.3</td>
<td>5.3</td>
<td>5.3</td>
<td>5.3</td>
<td>5.3</td>
</tr>
</tbody>
</table>

---

<table>
<thead>
<tr>
<th>Average Stall Per Frame (ms)</th>
<th>0</th>
<th>50</th>
<th>100</th>
<th>150</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decoder→Screen</td>
<td>-</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Loader→Decoder</td>
<td>-</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

**Note:**
- Request Forwarding (RF) reduces the exposed network traffic.
- DynaMix (DM) optimizes video quality for different resolutions.
Evaluation Setup: Smart Monitoring

Smart Monitoring

Master Device (Local)

Camera Device (Remote)

Android-based smartphones (5.1.1)
- Nexus 5 (master)
- Nexus 5 (camera) x 3

Selector

Processing

Screen

Camera

Camera

Phone (Master)

Phone

Camera

Phone

Nexus 5 (master)

Nexus 5 (camera) x 3
Throughput Improvement
- As the camera preview resolution increases

• Achieve higher throughput than Request Forwarding

![Throughput Improvement Graph](image)

Ideal (15 FPS)

- Request Forwarding
- DynaMix

3x higher
Computation Bottleneck

- Per-frame detection latency

**Potential to achieve higher throughput with faster CPUs**

![Graph showing avg. frame latency vs. preview resolution (pixels)]
Conclusion

- **DynaMix: Efficient cross-device resource sharing**
  - Transparent resource integration for diverse resources
  - Resource-aware dynamic task redistribution

- **Implementation on top of Android/Tizen devices**
  - Achieve target performance for multi-device services
  - e.g., 8.2 FPS → 24 FPS on the home theater scenario
DynaMix: Dynamic Mobile Device Integration for Efficient Cross-device Resource Sharing

Thank You!

Dongju Chae
torr55@postech.ac.kr
Department of Computer Science & Engineering
Pohang University of Science and Technology (POSTECH)