PALS ARCHITECTURE
(PHYSICALLY-ASYNCHRONOUS LOGICALLY-SYNCHRONOUS)

Cheolgi Kim
Multi-processing rather than multi-threading

- Failure of a function must not propagate to others
- c.f. processes with different criticality must reside in different VMs

- Message interleaving is one of main sources of complexity
MESSAGE INTERLEAVING

- A major contributor to No Fault Found problem, #1 complaint by airlines*
- Model checking state space grows exponentially due to message interleaving

SYNCHRONOUS DESIGN
VS
ASYNCHRONOUS DESIGN
IN DISTRIBUTED SOFTWARE
SYNCHRONOUS MODEL

- Lessons from H/W circuits
  - Nearly all digital circuits are synchronous
  - Synchronous model is proven to work
SYNCHRONOUS MODEL

- Computation is triggered by a clock tick at each round
- Computation changes the node’s state and issues messages to other nodes
- A message is destined at the next round
To compare synchronous and asynchronous design

Requirement

- One and only one side must be active, which is alive
- When toggle button is triggered, active side must switch to the other as long as both are alive
SYNCHRONOUS DESIGN FOR ACTIVE-STANDBY

Side1::eachPeriod() {
    if( side1 == side2 ) {
        side1 = ACTIVE;
    } else if ( side1 == NOT_ALIVE ) {
        side1 = STANDBY;
    } else if ( side2 == NOT_ALIVE ) {
        side1 = ACTIVE;
    } else if( toggle ) {
        side1 = flip( side1 );
    } else {
        side1 = side1;
    }
}

Side2::eachPeriod() {
    if( side1 == side2 ) {
        side2 = STANDBY;
    } else if ( side2 == NOT_ALIVE ) {
        side2 = STANDBY;
    } else if ( side1 == NOT_ALIVE ) {
        side2 = ACTIVE;
    } else if( toggle ) {
        side2 = flip( side2 );
    } else {
        side2 = side2;
    }
}
Asynchronous Model

- Each node has queues to hold messages
- Computation and communications have no restriction

Figure 7: Active-standby system configuration in an asynchronous environment

Our asynchronous design is based on [6], which presents the design and verification of an asynchronous active-standby system. The authors claimed that they performed model-checking to verify the correctness of the system, and it took 35 hours in NuSMV [5]. Moreover, the authors mentioned that the design took about 6 months to be fully completed and verified. Since the document does not include the precise design of the system, the details of our design may not be the same as [6]. Indeed, our current design is not complete with a proof of safety. This design has the following basic behaviors:

- An alive controller generates `EvtHeartbeat` to the other controller to notify its liveness.
- If no `EvtHeartbeat` comes in for a while, the controller regards the other side has failed, taking `ACTIVE` side role.
- The console delivers `EvtManualSelect` only to `Side1` at the event of user input. If `Side1` is not supposed to process the event, the event is forwarded to `Side2`. The reason why the event is forwarded by `Side1` rather than being broadcasted simultaneously to both controllers by the console is to avoid race condition.

- `(EvtManualSelect)` is only processed by the `STANDBY` controller. On the reception of the event, the `STANDBY` controller switches its role to `ACTIVE`, and enforces the other side to be `STANDBY` by sending `EvtStandby`.
DESIGN I – HEARTBEATS

- A node must exchange heartbeats to be a watchdog of each other
- No heartbeat reception => switch to active side

```c
void tx_timer_triggered()
{
    send_heartbeat( my_state );
}

void rx_timeout_triggered()
{
    my_state = ACTIVE;
}

void handle_heartbeat( int other_state )
{
    reset_rx_timeout();
    if( my_state == STANDBY && other_state == STANDBY )
    {
        my_state = ACTIVE;
        send_msg( EVT_BE_STANDBY );
        /* do nothing */
    }
}
```

side1

side2
DESIGN I – TOGGLE

- Standby side initiates toggle (NASA report)
  - Standby side switches to active, and ask the other to switch to standby
- Serialized event processing

```c
void handle_toggle()
{
    if( my_state == STANDBY ) {
        send_msg( EVT_BE_STANDBY );
        my_state = ACTIVE;
    }
}

void handle_be_standby()
{
    my_state = STANDBY;
}
```
PROBLEM OF DESIGN I

- Problem found by our model checking tool
- Delayed delivery of toggle message causes no toggle
- Side 1 relays **toggle** message to Side 2
- Total serialization of messages
- Hard to apply to triple redundancy – not scalable
YET ANOTHER PROBLEM

- Heartbeat msg can be interleaved
Lessons Learned

- Asynchronous design does not look simple even for very simple active-standby configuration.
- Some flaws in asynchronous design is not easily detected by code review.
- Code does not describe message interleaving.
PHYSICALLY
ASYNCHRONOUS
LOGICALLY
SYNCHRONOUS
(PALS)
SYSTEM
Largely distributed system cannot have physically global clock-tick generator for synchronous model

PALS realize logically synchronous system without a global clock generator

Better performance and semantics than TTA (Time Triggered Architecture)

Presented by TTTech at DASC 2011
PALS ARCHITECTURE

- PALS is designed to realize synchronous model of computation where there is no global clock generator.

- PALS parameters
  - Local time references have bounded jitter: $\epsilon$
  - Max computation time is given by $\alpha_{\text{max}}$
  - Max network delay is given by $\mu_{\text{max}}$
PALS OVERVIEW I

- All nodes have bounded jitters from global reference
- Every node triggers computation at same local time
- Round interval is given by
  \[ T \geq \mu_{\text{max}} + 2\epsilon + \max(\alpha_{\text{max}}, 2\epsilon) \]
- To deliver messages before next round of receiver
Message may be delivered to the same round

- Message must be sent after shaded time since all the tasks start within the time

![Diagram showing global reference of clock tick and tasks PALS task 1, task 2, task 3 with shaded time intervals and 2ε notation.](image)
REMAINED PROBLEM

- Message may be delivered to the same round

- Message must be sent after shaded time since all the tasks start within the time
PALS OVERVIEW II

- Receiver samples messages at each local clock tick
- Sender transmits messages with minimal delay from clock tick time of: \(2\epsilon\)
- Messages from round \(i\) are delivered to round \((i + 1)\)
Communications between tasks in different rates are performed in hyper period.

A sync thread running in hyper period is employed.

- The sync task must run first.
FLEXPALS

- Extension of PALS for Practice
  - More realistic implementation
  - Impose flexibility in synchrony
  - More scalable behavior in time
Problem of intentional tx delay of original PALS

Sampling can be also be delayed

Delayed transmission is hard to implement

In FlexPALS, each message has timestamp of clock-tick time, with which receiver resolves reception
FLEXPALS WITH PHASES

- A PALS period is sub-divided by phases
- For Lockheed Martin request

![Diagram of PALS phases](image)

- Decision logic
- Inverted Pendulum Subsystem (Task)
- Console Task
  - Pendulum Guidance System (PGS)
- HPC Task
  - High-Performance Controller (HPC)
  - High-Assurance Controller (HAC)
- HAC Task

Period boundary:

\[
t(P_i) = p_i \\
\]

Jitter tolerance:

\[
\begin{align*}
\phi_0(i) &< \phi_1(i) < \phi_2(i) < \phi_3(i) \\
\end{align*}
\]

Dec. logic:

\[
t(P_{i+1}) = t(P_i) + \text{jitter tolerance}
\]
PROBLEM OF MULTI-RATE PALS

- At hyper period, control tasks have delayed execution
- Performance is bounded by the worst case of jitters, computation time, and network delay
FLEXPALS WITH DELAYED EXECUTION

- No sync task is needed
- For scalability of multi-rate PALS, supervisory task execution may be delayed

\[ t(P_i) = p_i \]

HPC Task

High-Performance Controller (HPC)

HAC Task

High-Assurance Controller (HAC)

Decision logic

Inverted Pendulum Subsystem (Task)

Consoles Task

Pendulum Guidance System (PGS)

Inverted Pendulum Control System

HAC Task

Decision logic

Inverted Pendulum Subsystem (Task)

Consoles Task

Pendulum Guidance System (PGS)

Inverted Pendulum Control System

HAC Task

Decision logic

Inverted Pendulum Subsystem (Task)

Consoles Task

Pendulum Guidance System (PGS)

Inverted Pendulum Control System

HAC Task

Decision logic

Inverted Pendulum Subsystem (Task)
Design maximally reflected Lockheed Martin requests

- The only division that has been applied in pilot project by Lockheed Martin

E-mail from Lui Sha to the group

*The fact that PALS was being transitioned was critical to get our contract extended. And we should all thank Charlie for doing a wonderful job in working with LMC engineers. By Dec 1, we will be in the 4th year, we want to emphasize things that are easier to transition in THIS MEETING.*
MODEL CHECKING APPLICATIONS WITH PALS FRAMEWORK
MOTIVATION

- Lockheed Martin is highly interested in formal verification of S/W in source code level.
- Once message interleaving complexity is removed by synchronous model, verification based on model checking should be viable.
- AADL* is getting accepted by avionics industry, which can be used as requirements to check.

* a design language for avionics systems
VERIFICATION SYSTEM ARCHITECTURE

AADL Design

C Implementation

source analyzer

API traces

logic traces

KLEE

compliance verification engine

logic verification engine

WCET analyzer

ASIIST

schedulability analyzer

Maude
We assume PALS library is good

Source code analyzer

- Uses KLEE to have exhaustive execution traces
- KLEE was selected for MC/DC equivalence

Distributed behavior analyzer

- Implemented in Maude model checking language
WHAT IT VERIFIES

- Schedulability Analysis
- Compliance verification engine
  - Hint from Windows Driver Verifier
  - API usage compliance
  - C code – AADL design compliance
- Logic verification engine
  - Yet needed to be improved
MEASURED COMPLEXITY OF ACTIVE-STANDBY

<table>
<thead>
<tr>
<th>n=1</th>
<th>n=2</th>
<th>n=3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synchronous</td>
<td>39</td>
<td>10809</td>
</tr>
<tr>
<td>Asynchronous</td>
<td>56948</td>
<td>258346</td>
</tr>
</tbody>
</table>

* n = max. queue size
WHY I AM HERE

- PALS framework verification
  - Library cannot be verified through model-checking
  - Like seL4, distributed system framework may be formally verifiable with minimal assumptions
- Real-time functional language
  - Avionics system needs verification
  - Avionics system must be real-time
  - Avionics SW is usually simple
  - Functional language with limitation for real-time!
GAP BETWEEN MODEL AND IMPLEMENTATION

- Modeling of time
  - Global clock reference does not exist, neither does jitter
  - Primary clock server may be altered for failures
  - Jitter from global clock must be replaced by clock skews with each other
  - Local time speed is adjusted over time

\[ \geq \mu_{\text{max}} + \max(\alpha_{\text{max}}, 2\varepsilon) \]
CLOCK SYNCHRONIZER

- Key of time system
- Algorithms
  - Christian algorithm - easier to verify
  - Phased lock loop (PLL) - more suitable for avionics
    - Based on PID control idea
- Hybrid approach
THANK YOU
HOW TO USE KLEE

(a) activity diagram showing typical real-time application for D² RTS model

configure real-time thread
create thread
initialize thread
det_wait_schedule
periodic operation
finalize

(b) symbolic path exploration tree for verification environment
Kripke Structure from Klee