
MODULE *Config*

EXTENDS *Naturals, FiniteSets, Sequences, TLC*

Indicates that a configuration change is waiting to be applied to the network
CONSTANT *Pending*

Indicates that a configuration change is being applied to the network
CONSTANT *Applying*

Indicates that a configuration change has been applied to the network
CONSTANT *Complete*

Indicates that a configuration change was successful
CONSTANT *Succeeded*

Indicates that a configuration change failed
CONSTANT *Failed*

Indicates a change is a configuration
CONSTANT *Config*

Indicates a change is a rollback
CONSTANT *Revert*

Indicates a device is available
CONSTANT *Available*

Indicates a device is unavailable
CONSTANT *Unavailable*

Indicates that an error occurred when applying a change
CONSTANT *Error*

The set of all nodes
CONSTANT *Node*

The set of all devices
CONSTANT *Device*

An empty constant
CONSTANT *Nil*

Per-node election state
VARIABLE *leadership*

Per-node per-device election state
VARIABLE *mastership*

A sequence of network-wide configuration changes

Each change contains a record of 'changes' for each device
 VARIABLE *networkChange*

A record of sequences of device configuration changes
 Each sequence is a list of changes in the order in which they
 are to be applied to the device
 VARIABLE *deviceChange*

A record of sequences of pending configuration changes to each device.
 VARIABLE *deviceQueue*

A record of device configurations derived from configuration changes pushed
 to each device.
 VARIABLE *deviceConfig*

A record of device states - either *Available* or *Unavailable*
 VARIABLE *deviceState*

A count of leader changes to serve as a state constraint
 VARIABLE *electionCount*

A count of configuration changes to serve as a state constraint
 VARIABLE *configCount*

A count of device availability changes to serve as a state constraint
 VARIABLE *availabilityCount*

Node variables
 $nodeVars \triangleq \langle leadership, mastership \rangle$

Configuration variables
 $configVars \triangleq \langle networkChange, deviceChange \rangle$

Device variables
 $deviceVars \triangleq \langle deviceQueue, deviceConfig, deviceState \rangle$

State constraint variables
 $constraintVars \triangleq \langle electionCount, configCount, availabilityCount \rangle$
 $vars \triangleq \langle leadership, mastership, networkChange, deviceChange, deviceConfig \rangle$

The invariant asserts that any configuration applied to a device implies that all prior configurations of the same device have been applied to all associated devices.

$TypeInvariant \triangleq$
 $\wedge \forall d \in \text{DOMAIN } deviceConfig :$
 $deviceConfig[d] \neq 0 \Rightarrow$

$$\begin{aligned}
& \text{Cardinality}(\text{UNION } \{\{y \in \text{DOMAIN } \text{deviceChange}[x] : \\
& \quad \wedge \text{deviceChange}[x][y].\text{network} < \text{deviceConfig}[x] \\
& \quad \wedge \text{deviceChange}[x][y].\text{status} \neq \text{Complete}\} : \\
& \quad x \in \text{DOMAIN } \text{deviceChange}\}) = 0
\end{aligned}$$

This section models leader election for control loops and for devices. Leader election is modelled as a simple boolean indicating whether each node is the leader for the cluster and for each device. This model implies the ordering of leadership changes is irrelevant to the correctness of the spec.

Set the leader for node n to l
 $\text{SetNodeLeader}(n, l) \triangleq$
 $\wedge \text{leadership}' = [\text{leadership} \text{ EXCEPT } ![n] = n = l]$
 $\wedge \text{electionCount}' = \text{electionCount} + 1$
 $\wedge \text{UNCHANGED } \langle \text{mastership}, \text{configVars}, \text{deviceVars}, \text{configCount}, \text{availabilityCount} \rangle$

Set the master for device d on node n to l
 $\text{SetDeviceMaster}(n, d, l) \triangleq$
 $\wedge \text{mastership}' = [\text{mastership} \text{ EXCEPT } ![n] = [\text{mastership}[n] \text{ EXCEPT } ![d] = n = l]]$
 $\wedge \text{electionCount}' = \text{electionCount} + 1$
 $\wedge \text{UNCHANGED } \langle \text{leadership}, \text{configVars}, \text{deviceVars}, \text{configCount}, \text{availabilityCount} \rangle$

This section models the northbound *API* for the configuration service.

Enqueue network configuration change c
 $\text{Configure}(c) \triangleq$
 $\wedge \text{networkChange}' = \text{Append}(\text{networkChange}, [$
 $\quad \text{type} \quad \mapsto \text{Config},$
 $\quad \text{changes} \mapsto c,$
 $\quad \text{value} \quad \mapsto \text{Len}(\text{networkChange}),$
 $\quad \text{status} \mapsto \text{Pending},$
 $\quad \text{result} \mapsto \text{Nil})$
 $\wedge \text{configCount}' = \text{configCount} + 1$
 $\wedge \text{UNCHANGED } \langle \text{nodeVars}, \text{deviceChange}, \text{deviceVars}, \text{electionCount}, \text{availabilityCount} \rangle$

Roll back configuration change c
 $\text{Rollback}(c) \triangleq$
 $\wedge \text{networkChange}' = \text{Append}(\text{networkChange}, [$
 $\quad \text{type} \quad \mapsto \text{Revert},$
 $\quad \text{changes} \mapsto \text{networkChange}[c].\text{changes},$
 $\quad \text{value} \quad \mapsto c,$
 $\quad \text{status} \mapsto \text{Pending},$
 $\quad \text{result} \mapsto \text{Nil})$
 $\wedge \text{configCount}' = \text{configCount} + 1$
 $\wedge \text{UNCHANGED } \langle \text{nodeVars}, \text{deviceChange}, \text{deviceVars}, \text{electionCount}, \text{availabilityCount} \rangle$

This section models a configuration change scheduler. The role of the scheduler is to determine when network changes can be applied and enqueue the relevant changes for application by changing their status from *Pending* to *Applying*. The scheduler supports concurrent application of non-overlapping configuration changes (changes that do not impact intersecting sets of devices) by comparing *Pending* changes with *Applying* changes.

Return the set of all network changes prior to the given change
 $PriorNetworkChanges(c) \triangleq \{n \in \text{DOMAIN } networkChange : n < c\}$

Return the set of all completed device changes for network change c
 $NetworkCompletedChanges(c) \triangleq \{d \in \text{DOMAIN } networkChange[c].changes : \wedge Cardinality(\{x \in \text{DOMAIN } deviceChange[d] : deviceChange[d][x].network = c\}) \neq 0 \wedge deviceChange[d][\text{CHOOSE } x \in \text{DOMAIN } deviceChange[d] : deviceChange[d][x].network = c].status = Complete\}$

Return a boolean indicating whether all device changes are complete for the given network change
 $NetworkChangesComplete(c) \triangleq Cardinality(NetworkCompletedChanges(c)) = Cardinality(\text{DOMAIN } networkChange[c].changes)$

Return the set of all incomplete device changes prior to network change c
 $PriorIncompleteDevices(c) \triangleq \text{UNION } \{ \text{DOMAIN } networkChange[n].changes : n \in \{n \in PriorNetworkChanges(c) : \neg NetworkChangesComplete(n)\} \}$

Return the set of all devices configured by network change c
 $NetworkChangeDevices(c) \triangleq \text{DOMAIN } networkChange[c].changes$

Return a boolean indicating whether network change c can be applied
 A change can be applied if its devices do not intersect with past device changes that have not been applied
 $CanApply(c) \triangleq Cardinality(NetworkChangeDevices(c) \cap PriorIncompleteDevices(c)) = 0$

Node n handles a network configuration change event c
 $NetworkSchedulerNetworkChange(n, c) \triangleq \wedge leadership[n] = \text{TRUE} \wedge networkChange[c].status = \text{Pending} \wedge CanApply(c) \wedge networkChange' = [networkChange \text{ EXCEPT } ![c].status = \text{Applying}] \wedge \text{UNCHANGED } \langle nodeVars, deviceChange, deviceVars, constraintVars \rangle$

This section models the network-level change controller. The network control loop reacts to both network and device changes. The network controller runs on each node in the cluster, and the control loop can only be executed on a node that believes itself to be the leader. Note, however, that the model does not require a single leader.

When a network change is received:

- If the network change status is *Pending*
- Add device changes for each configured device
- If the network change status is *Applying*
- Update device change statuses to *Applying*

When a device change is received:

- If all device change statuses for the network are *Complete*
 - Mark the network change *Complete* with a *Succeeded* result if all device changes succeeded
 - Otherwise mark the network change *Complete* with a *Failed* result
- If all device changes failed due to *Unavailable* devices
 - Set the network change back to *Pending*
- If at least one device change succeeded but a subset failed
 - Fail the network change

Updates to network and device changes are atomic, and real-world implementations of the spec must provide for atomic updates for network and device changes as well. This can be done using either optimistic or pessimistic concurrency control.

Return a boolean indicating whether change c on device d already exists

$$\text{HasDeviceChange}(d, c) \triangleq \text{Cardinality}(\{x \in \text{DOMAIN } \text{deviceChange}[d] : \text{deviceChange}[d][x].\text{network} = c\}) \neq 0$$

Return the index of the device change for network change c

$$\text{DeviceChange}(d, c) \triangleq \text{CHOOSE } x \in \text{DOMAIN } \text{deviceChange}[d] : \text{deviceChange}[d][x].\text{network} = c$$

Return a boolean indicating whether the device change for network change c has status s

$$\text{HasDeviceStatus}(d, c, s) \triangleq \text{HasDeviceChange}(d, c) \wedge \text{deviceChange}[d][\text{DeviceChange}(d, c)].\text{status} = s$$

Add change c on device s

$$\begin{aligned} \text{AddDeviceChange}(d, c) \triangleq & \text{IF } d \in \text{DOMAIN } \text{networkChange}[c].\text{changes} \wedge \neg \text{HasDeviceChange}(d, c) \text{ THEN} \\ & \text{Append}(\text{deviceChange}[d], [\\ & \quad \text{network} \mapsto c, \\ & \quad \text{type} \mapsto \text{networkChange}[c].\text{type}, \\ & \quad \text{status} \mapsto \text{Pending}, \\ & \quad \text{value} \mapsto \text{networkChange}[c].\text{value}, \\ & \quad \text{result} \mapsto \text{Nil}, \\ & \quad \text{reason} \mapsto \text{Nil}]) \\ & \text{ELSE} \\ & \quad \text{deviceChange}[d] \end{aligned}$$

Change the status of change c on device s from *Pending* to *Applying*

$$\begin{aligned} \text{ApplyDeviceChange}(d, c) \triangleq & \text{IF } d \in \text{DOMAIN } \text{networkChange}[c].\text{changes} \text{ THEN} \\ & \text{IF } \text{HasDeviceChange}(d, c) \text{ THEN} \\ & \quad \text{IF } \text{HasDeviceStatus}(d, c, \text{Pending}) \text{ THEN} \\ & \quad \quad [\text{deviceChange}[d] \text{ EXCEPT } ![\text{DeviceChange}(d, c)].\text{status} = \text{Applying}] \\ & \quad \text{ELSE} \end{aligned}$$

```

    deviceChange[d]
ELSE
    Append(deviceChange[d], [
        network ↦ c,
        type ↦ networkChange[c].type,
        status ↦ Applying,
        value ↦ networkChange[c].value,
        result ↦ Nil,
        reason ↦ Nil])
ELSE
    deviceChange[d]

Change the status of change c on device s to Pending
PendDeviceChange(d, c) ≜
    IF d ∈ DOMAIN networkChange[c].changes THEN
        [deviceChange[d] EXCEPT ![DeviceChange(d, c)].status = Pending]
    ELSE
        deviceChange[d]

Return the set of all device changes for network change c
DeviceChanges(c) ≜
    {deviceChange[d][DeviceChange(d, c)] :
        d ∈ {d ∈ DOMAIN networkChange[c].changes : HasDeviceChange(d, c)}}

Return a boolean indicating whether all device changes for network change c are complete
DeviceChangesComplete(c) ≜
    Cardinality({x ∈ DeviceChanges(c) : x.status = Complete}) = Cardinality(DeviceChanges(c))

Return a boolean indicating whether all device changes for network change c were successful
DeviceChangesSucceeded(c) ≜
    Cardinality({x ∈ DeviceChanges(c) : x.result = Succeeded}) = Cardinality(DeviceChanges(c))

Return a boolean indicating whether all device changes failed due to Unavailable devices
DeviceChangesUnavailable(c) ≜
    Cardinality({x ∈ DeviceChanges(c) : x.result = Failed ∧ x.reason = Unavailable}) =
        Cardinality(DeviceChanges(c))

Node n handles a network configuration change c
NetworkControllerNetworkChange(n, c) ≜
    ∧ leadership[n] = TRUE
    ∧ LET change ≜ networkChange[c] IN
        ∨ ∧ change.status = Pending
            ∧ Cardinality({d ∈ DOMAIN networkChange[c].changes :
                ¬HasDeviceStatus(d, c, Pending)}) > 0
            ∧ deviceChange' = [ d ∈ Device ↦ AddDeviceChange(d, c) ]
        ∨ ∧ change.status = Applying
            ∧ Cardinality({d ∈ DOMAIN networkChange[c].changes :

```

$$\begin{aligned}
& \neg \text{HasDeviceStatus}(d, c, \text{Applying}) \} \} > 0 \\
& \wedge \text{deviceChange}' = [d \in \text{Device} \mapsto \text{ApplyDeviceChange}(d, c)] \\
& \wedge \text{UNCHANGED } \langle \text{nodeVars}, \text{networkChange}, \text{deviceVars}, \text{constraintVars} \rangle
\end{aligned}$$

Node n handles a device configuration change c
 $\text{NetworkControllerDeviceChange}(n, d, c) \triangleq$
 $\wedge \text{leadership}[n] = \text{TRUE}$
 $\wedge \text{LET } \text{change} \triangleq \text{deviceChange}[d][c]$
 IN
 $\wedge \text{change.status} = \text{Complete}$
 $\wedge \text{networkChange}[\text{change.network}].\text{status} \neq \text{Complete}$
 $\wedge \vee \wedge \text{DeviceChangesUnavailable}(\text{change.network})$
 $\wedge \text{networkChange}' = [\text{networkChange} \text{ EXCEPT } ![\text{change.network}] = [$
 $\text{networkChange}[\text{change.network}] \text{ EXCEPT}$
 $!.status = \text{Pending}]]$
 $\vee \wedge \text{DeviceChangesComplete}(\text{change.network})$
 $\wedge \text{DeviceChangesSucceeded}(\text{change.network})$
 $\wedge \text{networkChange}' = [\text{networkChange} \text{ EXCEPT } ![\text{change.network}] = [$
 $\text{networkChange}[\text{change.network}] \text{ EXCEPT}$
 $!.status = \text{Complete}, !.result = \text{Succeeded}]]$
 $\vee \wedge \text{DeviceChangesComplete}(\text{change.network})$
 $\wedge \neg \text{DeviceChangesSucceeded}(\text{change.network})$
 $\wedge \text{networkChange}' = [\text{networkChange} \text{ EXCEPT } ![\text{change.network}] = [$
 $\text{networkChange}[\text{change.network}] \text{ EXCEPT}$
 $!.status = \text{Complete}, !.result = \text{Failed}]]$
 $\wedge \text{UNCHANGED } \langle \text{nodeVars}, \text{deviceChange}, \text{deviceVars}, \text{constraintVars} \rangle$

This section models the device-level change controller. The device control loop reacts to device changes and applies changes to devices. The device controller runs on each node in the cluster. A master is elected for each device, and the control loop can only be executed on the master for the device targeted by a change. Note, however, that the model does not require a single master per device. Multiple masters may exist for a device without violating safety properties.

When a device change is received: - If the node believes itself to be the master for the device and the change status is *Applying*, apply the change

- Set the change status to *Complete*
- If the change was applied successfully, set the change result to *Succeeded*
- If the change failed, set the change result to *Failed*

Note: the above is modelled in two separate steps to allow the model checker to succeed and fail device changes.

Updates to network device changes are atomic, and real-world implementations of the spec must provide for atomic updates for network and device changes as well. This can be done using either optimistic or pessimistic concurrency control.

Node n handles a device configuration change event c
 $\text{DeviceControllerDeviceChange}(n, d, c) \triangleq$
 $\wedge \text{mastership}[n][d] = \text{TRUE}$

$\wedge \text{LET } change \triangleq deviceChange[d][c]$
 IN
 $\wedge change.status = Applying$
 $\wedge Cardinality(\{i \in \text{DOMAIN } deviceQueue[n][d] : deviceQueue[n][d][i] = c\}) = 0$
 $\wedge \vee \wedge change.type = Config$
 $\wedge deviceQueue' = [deviceQueue \text{ EXCEPT } ![n] = [$
 $\quad deviceQueue[n] \text{ EXCEPT } ![d] = Append(deviceQueue[n][d], c)]$
 $\vee \wedge change.type = Revert$
 $\wedge networkChange[deviceChange[change.value]].status = Complete$
 $\wedge networkChange[deviceChange[change.value]].result = Succeeded$
 $\wedge deviceQueue' = [deviceQueue \text{ EXCEPT } ![n] = [$
 $\quad deviceQueue[n] \text{ EXCEPT } ![d] = Append(deviceQueue[n][d], c)]$
 $\wedge \text{UNCHANGED } \langle nodeVars, configVars, deviceConfig, deviceState, constraintVars \rangle$

Return a sequence with the head removed
 $Pop(q) \triangleq SubSeq(q, 2, Len(q))$

Mark change c on device d succeeded
 $SucceedChange(n, d) \triangleq$
 $\wedge Len(deviceQueue[n][d]) > 0$
 $\wedge deviceState[d] = Available$
 $\wedge deviceChange' = [deviceChange \text{ EXCEPT } ![d] = [$
 $\quad deviceChange[d] \text{ EXCEPT } ![deviceQueue[n][d][1]] = [$
 $\quad deviceChange[d][deviceQueue[n][d][1]] \text{ EXCEPT }$
 $\quad \quad !.status = Complete,$
 $\quad \quad !.result = Succeeded]]]$
 $\wedge deviceConfig' = [deviceConfig \text{ EXCEPT } ![d] =$
 $\quad deviceChange[d][deviceQueue[n][d][1]].value]$
 $\wedge deviceQueue' = [deviceQueue \text{ EXCEPT } ![n] = [$
 $\quad deviceQueue[n] \text{ EXCEPT } ![d] = Pop(deviceQueue[n][d]))]$
 $\wedge \text{UNCHANGED } \langle nodeVars, networkChange, deviceState, constraintVars \rangle$

Mark change c on device d failed
 A change can be failed if the device rejects the change or the device is not reachable.
 $FailChange(n, d) \triangleq$
 $\wedge Len(deviceQueue[n][d]) > 0$
 $\wedge deviceChange' = [deviceChange \text{ EXCEPT } ![d] = [$
 $\quad deviceChange[d] \text{ EXCEPT } ![deviceQueue[n][d][1]] = [$
 $\quad deviceChange[d][deviceQueue[n][d][1]] \text{ EXCEPT }$
 $\quad \quad !.status = Complete,$
 $\quad \quad !.result = Failed]]]$
 $\wedge deviceQueue' = [deviceQueue \text{ EXCEPT } ![n] = [$
 $\quad deviceQueue[n] \text{ EXCEPT } ![d] = Pop(deviceQueue[n][d]))]$
 $\wedge \text{UNCHANGED } \langle nodeVars, networkChange, deviceConfig, deviceState, constraintVars \rangle$

This section models device states. Devices begin in the *Unavailable* state and can only be configured while in the *Available* state.

Set device d state to *Available*

$$\begin{aligned} \text{ActivateDevice}(d) &\triangleq \\ &\wedge \text{deviceState}' = [\text{deviceState} \text{ EXCEPT } ![d] = \text{Available}] \\ &\wedge \text{availabilityCount}' = \text{availabilityCount} + 1 \\ &\wedge \text{UNCHANGED } \langle \text{nodeVars}, \text{configVars}, \text{deviceQueue}, \text{deviceConfig}, \text{electionCount}, \text{configCount} \rangle \end{aligned}$$

Set device d state to *Unavailable*

$$\begin{aligned} \text{DeactivateDevice}(d) &\triangleq \\ &\wedge \text{deviceState}' = [\text{deviceState} \text{ EXCEPT } ![d] = \text{Unavailable}] \\ &\wedge \text{availabilityCount}' = \text{availabilityCount} + 1 \\ &\wedge \text{UNCHANGED } \langle \text{nodeVars}, \text{configVars}, \text{deviceQueue}, \text{deviceConfig}, \text{electionCount}, \text{configCount} \rangle \end{aligned}$$

Init and next state predicates

$$\begin{aligned} \text{Init} &\triangleq \\ &\wedge \text{leadership} = [n \in \text{Node} \mapsto \text{FALSE}] \\ &\wedge \text{mastership} = [n \in \text{Node} \mapsto [d \in \text{Device} \mapsto \text{FALSE}]] \\ &\wedge \text{networkChange} = \langle \rangle \\ &\wedge \text{deviceChange} = [d \in \text{Device} \mapsto \langle \rangle] \\ &\wedge \text{deviceQueue} = [n \in \text{Node} \mapsto [d \in \text{Device} \mapsto \langle \rangle]] \\ &\wedge \text{deviceConfig} = [d \in \text{Device} \mapsto 0] \\ &\wedge \text{deviceState} = [d \in \text{Device} \mapsto \text{Unavailable}] \\ &\wedge \text{electionCount} = 0 \\ &\wedge \text{configCount} = 0 \\ &\wedge \text{availabilityCount} = 0 \end{aligned}$$

$$\begin{aligned} \text{Next} &\triangleq \\ &\vee \exists d \in \text{SUBSET } \text{Device} : \\ &\quad \text{Configure}([x \in d \mapsto 1]) \\ &\vee \exists c \in \text{DOMAIN } \text{networkChange} : \\ &\quad \text{Rollback}(c) \\ &\vee \exists n \in \text{Node} : \\ &\quad \exists l \in \text{Node} : \\ &\quad \quad \text{SetNodeLeader}(n, l) \\ &\vee \exists n \in \text{Node} : \\ &\quad \exists d \in \text{Device} : \\ &\quad \exists l \in \text{Node} : \\ &\quad \quad \text{SetDeviceMaster}(n, d, l) \\ &\vee \exists n \in \text{Node} : \\ &\quad \exists c \in \text{DOMAIN } \text{networkChange} : \\ &\quad \quad \text{NetworkSchedulerNetworkChange}(n, c) \\ &\vee \exists n \in \text{Node} : \\ &\quad \exists c \in \text{DOMAIN } \text{networkChange} : \\ &\quad \quad \text{NetworkControllerNetworkChange}(n, c) \end{aligned}$$

$$\begin{aligned}
& \forall \exists n \in \text{Node} : \\
& \quad \exists d \in \text{Device} : \\
& \quad \quad \exists c \in \text{DOMAIN } \text{deviceChange}[d] : \\
& \quad \quad \quad \text{NetworkControllerDeviceChange}(n, d, c) \\
& \forall \exists n \in \text{Node} : \\
& \quad \exists d \in \text{Device} : \\
& \quad \quad \exists c \in \text{DOMAIN } \text{deviceChange}[d] : \\
& \quad \quad \quad \text{DeviceControllerDeviceChange}(n, d, c) \\
& \forall \exists n \in \text{Node} : \\
& \quad \exists d \in \text{Device} : \\
& \quad \quad \text{SucceedChange}(n, d) \\
& \forall \exists n \in \text{Node} : \\
& \quad \exists d \in \text{Device} : \\
& \quad \quad \text{FailChange}(n, d) \\
& \forall \exists d \in \text{Device} : \\
& \quad \text{ActivateDevice}(d) \\
& \forall \exists d \in \text{Device} : \\
& \quad \text{DeactivateDevice}(d) \\
& \text{Spec} \triangleq \text{Init} \wedge \Box[\text{Next}]_{\text{vars}}
\end{aligned}$$

\ * Modification History
\ * Last modified *Mon Sep 30 13:34:45 PDT 2019* by *jordanhalterman*
\ * Created *Fri Sep 27 13:14:24 PDT 2019* by *jordanhalterman*