Approaching to the high availability by using architectural concepts

Photograph of Presenter

M.A. Torkamani1, M.Yaghoobi2, S.Bashavard3
1Shahid Beheshti university, Tehran, Iran,
torkamani_ali@yahoo.com;
2Shahid Beheshti university, Tehran, Iran,
Mahmoud_yaghoobi@yahoo.com;
3Shahid Beheshti university, Tehran, Iran,
Samereh_bashavard@yahoo.com;

Paper Reference Number: 2612-619
Name of the Presenter: S.Bashavard

Abstract

The architecture of a software system has very significant role in its success. The architecture of software considers both its functional and none-functional requirements that are called quality attributes, too. One of the main qualities attributes that is very important to the stockholders of most of the information systems and is usually selected as the architectural driver is the availability. In this article we describe the method of approaching to the high availability by using the patterns and tactics of architecture. Indeed we show by accurate calculations that if the architect choose the proper patterns and tactics of the architecture, theoretically, he/she can reach to the availability of about 99%.

Key words: Software architecture, availability, quality attribute scenario, The Architectural patterns, The Architectural tactics

1. Introduction

Architecture of a program or computational system is structure(s) of system that shows visible properties from the outside of the elements and their relations [1]. The Architecture designing should be done in such a way that the system stockholder mentioned requirements be covered. One of the Architecture designing methods is attribute driven design. In this method, the main system is divided in to several subsystems and some of first surfaces of each subsystem and modules are obtained. In this method, we select few ones (less than ten [1]) of these system functional and non functional requirement that have higher priority than the rest, as architectural driver. The architect should choose some of architectural concepts to satisfy the architectural drivers. Architectural concepts are styles, patterns or tactics of the architecture. One of the main qualities attributes that is selected as driver in most of projects is availability. The availability means when we need the system, it is in operating state and it can provide the required service to us. In order to reach the high availability, the system should have high tolerance fault. In this article we want to talk about the way of reaching to the high availability using the architectural concepts. The article structure is as follows that we first describe the quality attribute scenario of fault tolerance in section 2. In the following (in section 3) we talk about the fault tolerance subsystem and describe the way of using Architecture patterns to reach the high availability. In last section we would have conclusions.

2. Quality scenario of fault tolerance

In software architecture, use case diagram represent the functional requirement and quality attribute scenarios is used to represent non-functional requirements [4]. Table 1 shows the scenario of fault tolerance. Table 2 shows quick recovery and table 3 show recovery scenarios. Table 4 shows also restart scenario. In order to reach and meet these scenarios, there are some architectural tactics that are shown in the table 5 [2]. In each project regarding the requirements and Constraints, special tactic(s) or pattern(s) is selected. The mentioned time in the scenarios (tables 1 to 5) is different from one system to the other one and are varying based the architect's experience. Meanwhile the architect should have come to an agreement on the quality scenarios with the Stockholders. In the following section of the article we will see that the architect can design subsystem of fault tolerance in a way that meet the mentioned scenarios be selecting the proper architecture pattern.

Table1: Fault Tolerance Quality Attribute Scenario

Scenario section	Possible value
Stimulus	Occurring a fail in the system
Stimulation resource	One of the components of software or hardware fails
Environment	System is servicing the users can currently
Artifact	Information system
Response	Recovery and restart are done (tables 3,4,5)
Response measure	Time of the system activation after failing

Table 2: Quick Recovery Quality Attribute Scenario

Scenario section	Possible value
Stimulus	Occurring a fail in the system
Stimulation resource	One of the components of software or hardware fails
Environment	System is servicing the users can currently
Artifact	Information system
Response	The presented responses to the users stop for maximum two second. This time is intangible for the user and he/she won't be aware of this issue.
Response measure	The secondary copy of the user request should operate after failing for maximum two seconds. All user requests that have been already sent and are close to failing time, should respond with maximum three seconds delay (on average)`

Table3: Slow Recovery Quality Attribute Scenario

Scenario section	Possible value
Stimulus	Occurring a fail in the system
Stimulation resource	One of the components of software or hardware fails
Environment	A spare processor is available that does not contain a copy of this component. A copy of the component is available on persistent storage and can be transferred to the spare processor via the LAN.
Artifact	Information system

Response	The users are informed that the service has become unavailable. A new copy of the service is started and becomes operational. The state of the component on restart may differ from that of the failed component but by no more than one minute. The users are then informed that the service is available for new queries.
response measure	The new copy is available within three minutes.

Table4: Restart Quality Attribute Scenario

Section of scenario	Possible values
Stimulus	A new replica is started is as stand by
Stimulation resource	The system resource manager starts the standby.
Environment	a failed has been occurred in system while it was servicing requests.
Artifact	Information system
Response	Starting a new copy in less than two seconds.
Response measure	Starting a new service would be done in less than two seconds.

Table 5: common tactics for availability

Fault detection	**Ping/echo:** One component issues a ping and expects to receive back an echo, within a predefined time, from the component under scrutiny. Lack of receiving the response is considered as fault. **Heartbeat:** One component emits a heartbeat message periodically and another component listens for it. If the heartbeat fails, the originating component is assumed to have failed and a fault correction component is notified.
Fault detection	**Active redundancy:** means additional components that are being replaced of failed ones, do exist. All redundant components respond to events in parallel. Consequently, they are all in the same state. The response from only one component is used (usually the first to respond), and the rest are discarded. **Passive redundancy:** One component (the primary) responds to events and informs the other components (the standbys) of state updates they must make. When a fault occurs, the system must first ensure that the backup state is sufficiently fresh before resuming services.
Fault prevention	**Removal from service:** This tactic removes a component of the system from operation to undergo some activities to prevent anticipated failures. **Process monitor:** Once a fault in a process has been detected, a monitoring process can delete the nonperforming process and create a new instance of it, initialized to some appropriate state as in the spare tactic.

3. fault tolerance Subsystem

As we said, one of the aims of architecture designing methods like attributes driven design (ADD) method is to decompose the original system and reach several initial surfaces of subsystems and modules .Regarding the importance of availability for the Stockholders, this quality attributes is usually selected as driver. To reach this quality attribute, most of the architects consider one subsystem for fault tolerance. As we said in the previous section, at first the quality attribute scenarios intended by the architect should be prepared by the system Stockholders. Then we should choose the proper architectural concepts for this subsystem. Table 6 shows the architectural concepts for subsystem of fault tolerance.

Table6: Architectural concepts for fault tolerance subsystem

Architectural concept	Child architecture concept	Section
Fault preparation	Restart	3.1
	Deployment	3.2
	Data integrity	3.3
Fault detection	Health monitoring	3.4
Fault recovery	Transparency to clients	3.5
	Replication	3.6
	Update client behavior after transient failure	3.7
	Update client behavior after hard failure	3.8

Before we discuss about each of the above mentioned items, it's necessary to indicate a point. In explaining each o f the architectural patterns (in tables 7 to 14) times should have been proposed. It's clear that determining these times would be based on the architect's experience. For example determining the period of Heart beat is done based on the experience and trade-off between Performance and availability that is described in the section related to the monitoring (section 3.4). Determining these time s is different from one system to the other one. Besides it, selecting proper software has also important role in these times. For example the more the elements of hardware are quick; the less the information inscribing on Log file. The numbers mentioned in this article are based on the experience and the research results in [2, 4] and have been just indicated as an example.

3.1 Restart

Two parameters exist for restart one failed component that include: Downtime Estimates and the service elimination (table 7).Usually selecting warm standby is a common selection. It may be proposed this question that why is warm standby being used in spite of the presence of load sharing pattern that has more speed. The reason of doing this task is trade-off between the performance and availability. In fact here, in order to choose a pattern, a c trade-off has been performed between the Performance and availability. That means instead of using a better pattern that has complexity and negative effect on performance, a lower pattern is used.

Table 7: Restarting parameters of a failed component

Row	Pattern	Downtime Estimates	Loss of service
1	Cold restart	Less than 2 minute	Yes
2	Warm standby	Less than 0.5 minute	Probably
3	Load sharing	Less than 0.05 minute	no

3.2 Deployment

In most of the projects, two main elements of the system are usually being considered to increase performance. This architectural pattern is called load balancing. In this pattern, a service is installed on several servers to decrease work load and so increase the performance. Consequently we would have two primary elements. For each of them, a secondary element is considered that is replication or copy of the primary. Here primary elements are denoted by A and B and the secondary ones with A' and B'. Checkpoint and save point of these elements are saved on the Persistent memory. Time of copy/ recovery of state for elements A and B are represented in table 8. These times have been considered based on experience.

The reason of difference between recovery times of these two elements could be hardware or software factors, etc. State recovery time of component a denoted by TrA and State recovery time of component B denoted by TrB:
TrA=0.8 sec (1)
TrB=0.6 sec (2)
In general there are two scenarios to the component deployment (Table 9). In order to reach high availability, the second pattern is suggested.

Table8: Time of copy/ recovery of state for elements A and B

Row	Component	Time (second)
1	A	0.8
2	B	2

Table9: component deployment scenarios

Row	Pattern	First processor	Second processor	A failure	State recovery time
1	One processor	A,B	A',B'	A',B'	2sec
2	separate	A,B	A,B'	A',B	0.5sec

3.3 Data integrity

Table 10 shows the architecture pattern related to data integrity.

Table 10: the architectural patterns related to Data integrity.

Row	Pattern	Communication loading	Stand y processor loading
1	Slow checkpoint	1.2 second per 1 minute	No
2	Fast checkpoint	1.2 second per2 second	No
3	Checkpoint + Log changes	1.2second per 1 minute +100messages in 1sec	no
4	Checkpoint +synchronize primary and backup	1.2 second per 1 minute +1 message per X sec	It implements to keep updated the state of a copy
5	Checkpoint+ bundled Log changes	1.2 second per 1 minute +1message per X second	no

Pattern 1 does not satisfy quick recovery for the status that had at least one minute lifetime. Pattern 2 has one unacceptable load in the communication system. Pattern 3 places a significant burden on the communication system. Pattern 4 is more complex, since the secondary must execute every *x* seconds to update its copy of the state. About pattern 5, if X be less than 2 second, quick and slow recoveries (table3, 4) do become satisfied. So the quality scenario 5 is mentioned as a proper selection. We represent reserve state of Log file by Tps.
Tps=2sec (3)

3.4 Health monitoring

Table 11 shows the candidate architectural patterns related to the health monitoring.

Table 11: candidate health monitoring architectural patterns

Row	Pattern	The communication loading
1	Heartbeat	4 messages (for A,B,A',B')
2	Ping/Echo	8 messages (Ping, echo for A,B,A',B')

According to table 11, ping/Echo needs 8 messages, while heartbeat just needs 4 messages. Therefore heartbeat is a proper selection.
Period of Heartbeat is considered 0.25 second. This time is shown by Th.
Th=0.25 sec (4)
Of course selecting heartbeat is determined by trade-off between the availability and performance. The less heartbeat, the more the number of sent messages and consequently the occupied bandwidth would be increased and the performance be decreased and the more the heartbeat ,the more the time duration that a system would be aware of the other system state via this and in the case of the second system failing or changing , the first one would be aware by more delay. So the value of Heartbeat could be determined by experience and it could not be determined in the same way for all the systems. Figure 1 shows the sequence diagram of A failing and A' promotion from secondary to primary. Figure 2 shows timing diagram related to the system operation while failing .Figure 3 shows the activity diagram of system fail and recovery.

3.5 *Transparency to clients*

Table 13 shows the candidate architectural patterns related to transparency. Regarding to table 12 pattern 1 has no transparency .We suggest using second pattern. When we use pattern 2, the proxy responsibility is that it should have the least fail time while transferring from primary system to secondary one. In section G, we would give more explanation in this regard.

Table 12 : candidate architectural patterns related to transparency.

Row	Pattern	Place	Transparency
1	Client handles failure	Client	No
2	Proxy handles failure	Proxy	yes
3	System handles failure	System	yes

3.6 Replication

The replicated elements provide transparency for the user. We denote the primary process by A and the secondary one by B. The replicated processes are denoted by A', B' replication is needed when a fault appears, based on this the candidate architectural patterns related to replication are shown in table 13. We suggest the primary replication. The replication increases the resource availability and also causes delays to decrease and help to increase performance.

Table 13: candidate architectural patterns related to replication

Row	Pattern	Process	Replicated process	Time
1	Primary replication	A	A'	Less than 0.5sec
2	Secondary replication	B	B'	Less than 2 min

3.7 Update client behavior after transient failure

This pattern is updating client after transient failure. The operation of the proxy service when a transient failure occurs has already been defined: The health monitor informs the proxy service of the failure. Then, this service sends a new secondary access code to each asynchronous communication mechanism. This access code will be used for the next update request. Essentially, this mechanism promotes the secondary to the primary. Figure 4 shows the primary presentation of the architecture related to the system failure time.

3.1 upgrade client behavior after hard failure

When a primary fails and no secondary is available, one of the design patterns in Table 14 could be used.

Table14: Patterns for Update Client Behavior after a Hard Failure

Row	Pattern	Result
1	Continue sending data	Unusable data is sent to the proxy and client
2	Stop sending data	The communication during downtime is saved to a log
3	Load Log data	The communication log is loaded

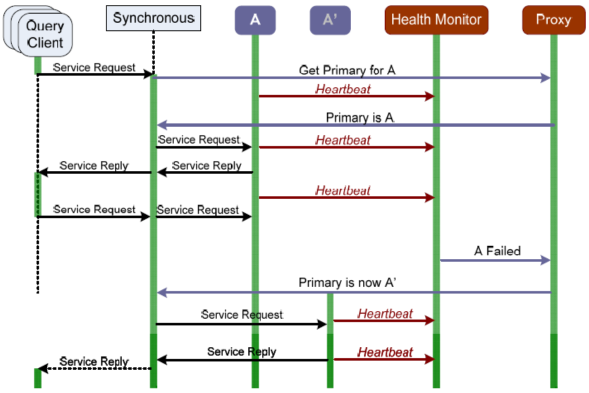

Fig 1: A Sequence Diagram of Failover from A to A'

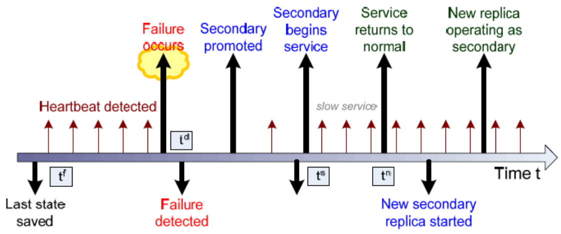

Fig 2: Timing Model related to the system operation while failing

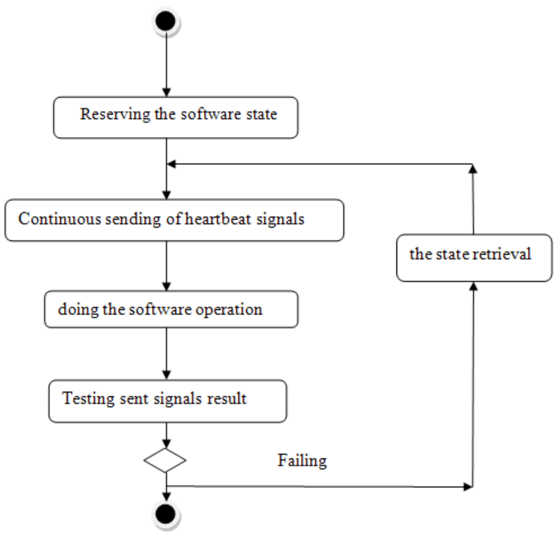

Fig 3: activity diagram of system fail and recovery

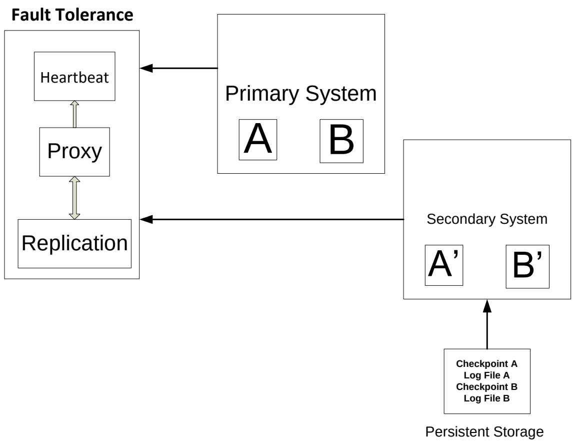

Fig 4: primary presentation of the architecture related to the

Using the first pattern does not seem reasonable that much. Pattern 3 is complex, so we ignore it. Therefore we use second pattern. That means when the system fails, we stop sending data until the secondary be available [4].

3.2 Calculating non-available time

In this section we want to examine that if the architect selects our suggested patterns and tactics, how much the system non-available time would be. Table 15 shows selected architectural patterns. Regarding this time in next section, we calculate availability value.

T1=Tps + TrA + TrB+ Th+TrL+Tus

We have already explained Th,TrB,TrA,Tps. Here we introduce two other parameter:
Trl: time needed to repair log file of persistent memory
Tus: time needed to update A, B states in log file
Trl and Tus are approximated based on experience as follows:
Trl= 0.2 sec (5)
Tus= 0.1 sec (6)

Table 15: Selected architectural patterns

Row	Architectural concept	Selected pattern	considerations
1	Warm Standby	Restart	row 2 of table 8
2	Separated processors	Deployment	row 2 of table 10
3	Checkpoint+ Bundled Log Changes	Data integrity	row 2 of table 11
4	Heartbeat	Health monitoring	row 2 of table 12
5	Proxy Handles Failure	Transparency to clients	row 2 of table 13
6	Main iteration	Replication	row 2 of table 14
7	Stop Sending Data	Update client behavior after hard failure	row 2 of table 15

In the worse state, failure might occur exactly after heartbeat. In this case regarding the relations of 1, 2, 3, 4, 5, 6, the system non-available time is equals to:
=2+0.8+0.6+0.25+0.2+0.1=3.95 sec (7)

3.3 Calculating availability

In [3] the below formula is represented to calculate availability:
A=Uptime/total time required to be operational (10)
In this relation, up time of the system available time and total time required to be operational is the time that we need the system and should be available. Suppose that our system fails once per 5 min. In this case every 5 min, the system would be available for 3.95sec. As a result, the availability equals:
Up time= 5*60sec -3.95 sec=296.05sec
A=296.05/300=0.98333333333333333333333333333333333 (8)
Therefore the system has 98.68% of availability

4. Conclusion

We have said that availability is one of quality attributes that is selected as architectural driver. To reach availability, the software architecture should be in such a way that this quality attribute be satisfied. In this article we represented the method of reaching to availability using tactics, patterns and the other architecture concepts. We saw that how we can reach the availability above 90%. The main point that we should consider is that some of quality attributes effect on each other negatively and have so –called negative correlation with the other quality attributes like availability. In such cases they should perform a trade-off among quality attributes (positive or negative relationship).

References

1- Bass, L. & Clements, P. & Kazman, R.(2003), Software Architecture in Practice, Second Edition, Addison Wesley.

2- GAMMA-J , GAMMA-J USB Web Store, "Attribute Driven Design (ADD) Document", http://avinator.com/GAMMA-J_USB_Web_Store_ADD.pdf
3- QuEST Forum (2001). TL 9000 Quality Management System Measurements Handbook, Release 3.
4- Wood, W.G. (2007). "A Practical Example of Applying Attribute-Driven Design (ADD) Version 2.0", SEI.