


Performance evaluation of redundant OPC UA architecture for process control

Transactions of the Institute of
Measurement and Control
2017, Vol. 39(3) 334–343
© The Author(s) 2017
Reprints and permissions:
sagepub.co.uk/journalsPermissions.nav
DOI: 10.1177/0142331215603792
journals.sagepub.com/home/tim


Rafal Cupek¹, Kamil Folkert¹, Marcin Fojcik², Tomasz Kłopot¹ and
Grzegorz Polaków¹

Abstract

Classical control applications with a centralized logic and distributed input/output system are being replaced by dynamic environments of cooperating components. Thus, the OPC (Object Linking and Embedding for Process Control) UA (Unified Architecture) is becoming more popular, because the OPC Data Access substandard is not well suited for distributed systems. Moreover, in many production systems, redundant data servers are preferred, for financial and legal reasons. Providing performance evaluation gives an estimate of the time required (and data samples lost) to switch to a backup data source for redundant OPC UA architecture, depending on the failure detection method, number of variables and redundancy mode.

Keywords

Fault detection/diagnosis, process control, process monitoring, redundancy, reliability

Introduction

OPC (formerly Object Linking and Embedding for Process Control) is nowadays widespread in industry. However, because the classical DCOM/COM (Distributed Component Object Model/Component Object Model)-based OPC substandard (nowadays known as OPC Classic) is not well suited for distributed systems, the OPC Unified Architecture (UA) is becoming more popular. At the same time, in many production systems, the demanded level of reliability leads to implementation of redundant data servers. Unfortunately, both the popular OPC Classic and the distributed UA mechanisms have not been adequately tested in regard to redundancy. Thus, process engineers are reluctant to look at these solutions. The analysis of the recovery time in redundant OPC servers can significantly assist engineers in an informed choice of OPC redundancy work mode. As a result, proper assessment of the timings, taking both the qualitative and quantitative criteria into the account, may lead to an increase in use of the OPC redundancy in process engineering applications, thus improving the financial outcome of respective businesses, as shown in the examples described in the following paragraphs. Proper functioning of communication is particularly important because the disturbances introduced by communicational errors can undermine the quality control improvement introduced by even the most advanced control algorithms (Kasprzyczak and Macha, 2015; Stebel et al., 2014).

There are good examples of process data importance in the automotive industry. The cables used in electrical harnesses in industry are subject to continuous quality control. Recipients of the product define the length of the segment, at the end of

which the measurement is made. As the test sections are short and the production process is fast, process logs are of large volume. The cable manufacturer cannot sell the product if there is no confirmation of product quality (cable parameters) in the form of logs collected at that stage of production.

During the production of airbags, the operator of the system obtains information about the type of the specific airbag and based on this information performs specific operations. The data on each of the airbags are stored in an radio-frequency identification (RFID) chip, which is read by a programmable logic controller (PLC). Next, the data are archived in a storage system, and the archive is the data source for the operator panel and the operator. Communication between the PLC and the data storage is performed with the OPC. Should the data link fail, the operator would have no knowledge of the operations expected to be performed, which would result in the production line being stopped. Moreover, because the airbags are essential for the safety of passengers in the car, the data on each piece are kept. The loss of the stored data could result in serious legal consequences.

Nowadays, in general in the automotive industry, each car is manufactured according to a specific customer order generated in the manufacturing execution system. Information on a particular car is passed along with it from the beginning,

¹Silesian University of Technology, Gliwice, Poland

²Høgskulen i Sogn og Fjordane, Sogndal, Førde, Norway

Corresponding author:

Grzegorz Polaków, Silesian University of Technology, Akademicka 16,
Gliwice, 44-100, Poland.

Email: grzegorz.polakow@polsl.pl

i.e. from welding – it is stored in a chip attached to a sled on which the car body is mounted. Problems with reading, transferring or storing the data will result in the production line being stopped. Modern production lines in the automotive industry work according to a very strictly defined schedule, so every disturbance in RFID reading desynchronizes the line, affecting economic performance.

Another example showing the importance of measurement data for economic performance of the company is the system of electricity contracting. The energy consumed by the entire industrial plant is anticipated and a contract is signed with the electricity provider for the predicted amount. When not all the contracted energy is used up or, conversely, the consumption exceeds the contracted amount, a financial penalty has to be paid. The penalties are severe to the extent that it is economically viable to order somewhat larger quantities of energy and potentially launch additional electrical receivers at the end of the trading period to fill the deal and avoid penalties. At present, the trading period is long enough that the fulfilment of an electricity contract by manipulating electrical loads can be managed manually. However, there are reasons to change the energy demand modelling interval number to 96 a day, which would make the contracting period 15 min long. The potential change is supported by the specificity of consumers and producers of electricity in the distribution network (Baczyński, 2008). With the new approach to the problem, introducing the redundancy mechanism to transmitting the data to the MES system can be economically justified. Due to the continuous tracking of energy consumption at the supervisory control level, a decision-making system can turn on or turn off devices that are not essential for the technological process to fill the energy contract. Therefore, the degree to which the energy contract will be filled depends directly on the data read from servers, and when the data is unavailable, the management system will be unable to track the power usage, which will most likely result in penalties.

The above examples refer to stopping the production lines or incurring direct financial losses in a result of data servers being incapable of providing the information on time. The goal is therefore to ensure that the process data servers will be available and ready to fulfil requests at any given moment. A term of availability can be introduced, which in this context is the ratio of fulfilled requests to all the requests issued, and it is shown in the above examples that many industrial applications rely on the high availability of OPC servers. One of the means of increasing the availability is redundancy, which, applied to the OPC UA technology, is investigated in this paper.

State of the art

SOA (service-oriented architecture)

The area of industrial networked systems is currently changing significantly. Classical distributed monolithic applications with centralized control logic and a distributed input/output system are being replaced by dynamic and holistic environments created by cooperating components. Industrial control systems based on SOA [e.g. SIRENA (Jammes and Smit, 2005) and SOCRADES (Moreira et al., 2008) projects] create

a highly flexible service-oriented infrastructure with more capabilities. Moreover, integrated access to process data on the field device level, access to events generated by control devices and direct access to process history data are important factors, which favour the object-oriented approach in contrast to a classic tag-oriented one (Cupek et al., 2009).

In the systems built according to the SOA principle, the services are defined independent of the hardware. On the one hand, this allows more flexibility in the use of redundant communication services (as independent services) and, on the other hand, in real-time systems the new service will require additional resources (processor, memory, network bandwidth), which may affect the performance of other communications services. Redundancy in SOA introduced at the application level may affect the performance of other services in the real-time system so its use can be more complicated than, for example, for the case of hardware medium redundancy or communication processor redundancy.

One of the main obstacles limiting the applications of the SOA approach in distributed industrial control systems area is the required change of the underlying communication model. SOAs cannot be developed on top of classical fieldbus networks defined in IEC 61158 (including standard communication profiles applications according to IEC 61784; Felser, 2005). This is because the classical control systems based on a PLC and distributed inputs/outputs operate on a shared memory model, i.e. process variables are cyclically exchanged between PLC memory and cooperating I/O devices. Such the system structure requires detailed scheduling of the communication at the stage of the system design.

The SOA paradigm is based on a message passing model. It has to be noted that not only the message delivery time but also a delivery order is crucial for control system behaviour. An example of a new architecture developed to fulfil these new requirements is the IEC 61499 standard (Vyatkin, 2011).

OPC UA

OPC UA is the first service-based communication standard, which was widely accepted by industry, and became an IEC 62541 standard in 2012 (see Cavalieri and Chiacchio, 2013, for a good introduction). The OPC UA services are defined on an abstract level and are implemented with a binary UA TCP encoding or as an XML encoding within the SOAP/HTTP transport protocol, which is a new approach when compared with the proprietary DCOM used in OPC Classic. Thus, the new architecture can be usable as a gateway for the increasingly popular idea of Internet of Things (see Min et al., 2014, for a similar discussion). Information exchange security is managed by a Secure Channel Service Set, which defines a long-running logical connection between an OPC UA client and server. This channel maintains a Public Key Infrastructure (PKI) that is used to authenticate and encrypt *Messages* sent across the network. Next, by means of a communication channel implemented by the OPC UA, the *Services* are provided. Thus, there is no need for tunnelling of the traffic through an additional secure layer, as was in case in the OPC data access standard.

Data access mechanisms provided by OPC UA include the classical on-demand client-server mode dubbed *Read/Write* services, which are defined in the *Attribute* service set. The mechanism allows clients to access the process data directly; however, a more efficient communication can be established on basis of a well-known *Publisher/Subscriber* model supported by *MonitorItem* and *Subscription* service sets. In this model, a client subscribes to server variables with given Node parameters: Sampling Interval (ms), Queue Size and Filter Conditions. The latter define Trigger conditions (status, value/status, source timestamp/value/status) and Deadband (Absolute, Percent). According to the defined tracking conditions, an OPC UA server sends a client notification for event detection in *MonitoredItem*. Data transfers are optimized by grouping within sessions for efficiency reasons. Published request responses are not sent immediately, but are queued by the server (Mahnke et al., 2009). The responses (*NotificationMessages*) are sent back according to the Subscription's publishing interval. The *NotificationMessages* contain either Notifications of *MonitoredItems* (when there is an event to be reported to the client) or a *KeepAlive* message (a dummy message to notify the client that the server is operational). The *KeepAlive* message is the new feature introduced in OPC UA, which is crucial for the 'watchdog' implementation of detection of OPC server failures.

OPC UA provides communication services but it relies on the delivery guarantees of an underlying communication network (see Cavalieri and Chiacchio, 2013, for an OPC UA performance analysis). Ensuring the communication reliability remains out of the scope of the OPC UA interface standard. There are many methodologies for fault detection of the control loop (Alkaya and Grimble, 2014; Frank et al., 2000; Patton et al., 1995) or fault tolerance for communication systems (Du et al., 2013; Yu et al., 2013). One of the strategies for providing the fault recovery capability is redundancy, i.e. duplication of the critical system's components to increase the system's general availability. Redundant solutions are widely used in real-time industrial computer systems to increase the reliability of their control and safety functions. The approaches to redundancy in industrial communication area include, amongst others, industrial network media redundancy (Felser, 2008; Gessner et al., 2013), Time- and Event-Triggered traffic methods, or master replication and fail-silence enforcement (Ferreira et al., 2006).

Redundancy in OPC UA

Although subscriptions are supported by session mechanisms that ensure reliable communication in case of transmission errors, the server or client fault-down errors may be solved by client or server redundancy only. OPC UA clients can discover server support redundancy mechanisms by dedicated data types and client server profiles. Because it is unnecessary to support redundancy in all OPC UA servers, this capability is described in a server and client profile definition. Profiles define complete functional units supported by given OPC UA servers or clients. Profiles are composed from Facets, which describe the elemental functionalities. The redundancy capability is defined in a Redundancy Conformance Group.

Redundancy could be used for high availability, fault tolerance and load balancing (OPC Foundation, 2010). Facets define redundancy properties for clients and servers, both with and without transparent redundancy support. Additional information is exposed to clients by servers with usage of *ServerRedundancyType*, *TransparentRedundancyType* and *NonTransparent-RedundancyType* variables (OPC Foundation, 2009).

The described properties of OPC UA SOA simplify communication redundancy creation in the following ways: 1) there is no longer a need for separate redundancy solutions for a given kind of application because one consistent *AddressSpace* and one SOA system model are defined; 2) the session-based communication supports redundancy by session transfer; and 3) the information about a supported redundancy model is available at the server.

In this paper, a server fault-down scenario and redundancy at the server level are considered. To create redundant solutions without a particular server or client modification, a Recovery Proxy component has to be added to the communication system, which may be done for the client (Client Recovery Proxy) or the server (Server Recovery Proxy). The proxy can be based on a transparent or non-transparent redundancy solution (Lange et al., 2010):

- *Transparent redundancy*: servers are responsible for synchronization and subscription management. Clients do not participate in this process, although the information about active and backup servers is available for clients. Although the server takes responsibility for all the failure-safe actions, the redundancy implementation is server provider dependent and redundancy mode cannot be adjusted to application needs. To avoid the dependency on a specific OPC UA server implementation, the non-transparent redundancy has to be used.
- *Non-transparent redundancy*: the client's proxy is responsible for checking both the main and backup server status. OPC UA non-transparent redundancy mode is based on the *TransferSubscription* subservice, which allows moving subscriptions between sessions. Such an approach allows for the movement of *MonitoredItems* from one session to another. Because the recovery proxy part is implemented in the client, the available number of specific implementations of data sources is greater than in the case of transparent redundancy (the server side does not require any modification, any plain server is supported). Three non-transparent redundancy scenarios are examined by the authors: 1) a COLD mode when all communication activities related to connecting, subscribing and sampling are started at the backup server after the main server failure; 2) a HOT mode in which both the main and backup servers are active and connected to the data source and only the publishing action requires switching; and 3) a WARM mode in which both the data sampling, and data publishing are activated at the backup server after the main server fault.

Table 1. Client's recovery modes (Chen and Bastani 1994).

Recovery mode	COLD	WARM	HOT
On initial connection:			
Connect to more than one OPC UA server		X	X
Create subscriptions and add monitored items		X	X
Activate sampling on the subscriptions			X
During recovery:			
Connect to backup OPC UA server	X		
Create subscriptions and add monitored items	X		
Activate sampling on the subscriptions	X	X	
Activate publishing	X	X	X

OPC, Object Linking and Embedding for Process Control; UA, Unified Architecture.

The order of server switching activities for the analysed modes is presented in Table 1.

The redundancy scenarios are proposed by Lange et al. (2010), and are specifically designed for distributed SOA-based systems in contrast to the scenarios known in real-time systems (Chen and Bastani, 1994).

The COLD standby does not age, so the error probability does not increase, as in other modes. Moreover, the influence of a backup server on the source of the information is minimized during the primary server activity. The price for this is the relatively high cost of the connection establishing after the main server failure. In the HOT mode, the goal is the reduction of the recovery time, for the cost of constant maintaining of the second data stream. The WARM mode seems to be a trade-off solution. Redundant server–client connections are created, but the redundant server does not collect data from the source. During the recovery, the backup server is activated and the subscriptions are transferred.

There are also additional reasons behind the WARM redundancy mode. The main motivation for the WARM mode in Chen and Bastani (1994) was conserving the processing resources and available memory. This results from the usual implementation of non-distributed redundant systems, where all the threads and processing take place in the same hardware resource. However, OPC servers do not use the same processor and memory pool, so the reason is invalid. The justification for the existence of the WARM mode is that in the HOT mode, both primary and backup OPC servers are connected to the same data source, and the controlled process is sampled by both of them simultaneously. So the WARM mode limits excessive source data access attempts, eliminating such issues as reaching the communication protocol's limits of the maximum allowed number of active connections to a data source, or degrading source device parameters (especially its time characteristics) by overusing the limited resources of a data source device.

It also should be noted that applying the HOT redundancy mode is physically impossible in many real-world cases. Sometimes communication protocol does not allow connection to more than one master, and sometimes more than one communication channel uses too many resources and causes freezes at the data source side. The WARM redundancy mode overcomes the limitations, because it creates redundant

communication channels at the OPC server side, but at the same time uses only one connection to the data source.

Materials and methods

Redundancy quality indicators

There are known methods for the mathematical calculation of the main quality parameter, i.e. reliability, knowledge of which is necessary in production planning. Measuring and comparing different redundancy scenarios and cases requires a measure of redundancy quality. The measure should take into account all the factors that affect the quality of the redundant systems, e.g. recovery duration, mean time between failures, redundancy type, amount of redundant devices, etc. This subsection provides theoretical calculations related to reliability, which lead to the specification of the redundancy quality measure used later in the experiments.

Known redundancy methods (Avizienis, 1995) are divided into two groups: static and dynamic. N-Version Programming (NVP) belongs to the static methods, and requires n ($n \geq 2$) different elements that constantly work and a method of element selection. The dynamic methods are divided into backward error recovery and forward error recovery. An example of the last one is the Recovery Block (RB), the principle of which is to detect a failure and then to perform an alternative action to correct the failure, i.e. to recover. Two examples that can be given are: exception processing and switching to a new data source if the active source is not responding.

From the engineer's point of view, the most important issue is finding how much data (information) is lost during the recovery. For now, no mathematical dependency between the recovery time and amount of lost data in OPC UA systems is known. The only method to verify this is to conduct tests in a real environment.

Redundancy in OPC UA is an instance of the Recovery Block approach. For that reason, it can be assumed that the total recovery time is calculated as:

$$t_R = t_D + t_A \quad (1)$$

where t_R is the total recovery duration, t_D the time required for failure detection and t_A the time required for recovery actions.

The failure detection time t_D describes how quickly the failure can be detected. This detection of server failure in OPC UA can be realized in various ways, but each is actually a variant of watchdog implementation. The client expects the response from server in a certain period after the request, and when the answer does not arrive in a preset time, a timeout exception is raised. Obviously, the watchdog approach is very simple and does not cover the cases of the data delivered on time but with wrong values, which leaves the approach with the diagnostic coverage parameter far below 100%. The wider approach should be considered in production environments, e.g. 2oo3 or Triple Modular Redundancy methods, but this falls out of the scope of the paper, which deals only with the server failures (e.g. because of overload), assuming that the communication links are reliable.

The possible implementation could be based on one of the following three scenarios:

- 1) asynchronous read operation (calling *Read* service of the server);
- 2) polling server status with SDK-based listener support,
- 3) subscription-based *KeepAlive* method (server must send the information with new data or a heartbeat).

The duration of recovery actions t_A is expected to remain the same between scenarios, as it depends solely on actions taken after the failure detection. However, the failure detection time t_D is expected to depend on the amount of data and redundancy scenario.

Experimental set-up

The redundant OPC system implemented for the experimentation is based on a Client–Server SDK by Prosys, which is written in a Java programming language. The SDK is backed up with the OPC Foundation's UA communication stack. The SDK utilizes UA communication, thus providing high-level interfaces for the developer. Because of this, the implementation of the redundant OPC UA communication needs far less engineering effort, and a developer can focus on a final product functionality instead of low-level communication maintenance and data exchange handling. The Java OPC UA SDK mentioned above proved its usability in the previous work of the authors on OPC UA efficiency tests (Folkert et al., 2011).

Java technology was chosen for the implementation due to its platform independence and portability. These features strongly correspond with OPC ideas of platform-independent, reliable communication in heterogeneous computer systems. Java applications may be deployed and run on various types of devices, including industrial and personal computers that are often used in the industrial automation environments. For that reason, Java seems to be a good base for OPC UA applications, as well as for research purposes.

As previously mentioned, the authors' implementation covers non-transparent server redundancy with the client's proxy. The research environment was based on several separate PCs (Windows 7, CPU Pentium 8400 3.2 GHz, 4 GB RAM; for additional tests Celeron 697 MHz, 320 MB RAM

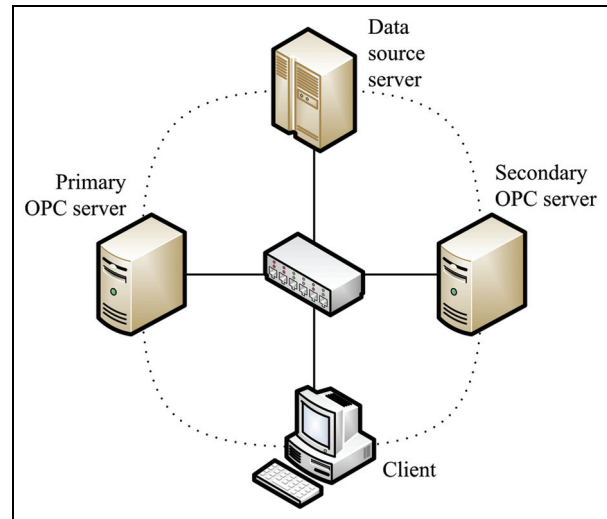


Figure 1. The experimental set-up (dotted lines – logical connections; continuous lines – physical connections).

was used as the redundant server), each responsible for handling execution of one system part application: two OPC UA servers (primary and secondary), OPC UA client and data source application. The Java runtime used was JRE 7u1 by Oracle. The communication between the data source application and the UA servers was based on low-level TCP sockets. All communication channels (UA, data source and synchronization) were established on a 100-Mbit FastEthernet local area network, which seems to be a credible substitute for an enterprise network (Figure 1).

Experiments and results

The decisive issue for considering reliability and amount of lost data in the experiment was the time parameters. In the research, the authors focused on fast data exchange processes, as they seem to be crucial in many systems; therefore, the redundancy is highly approved to be used in such cases. The timings were measured depending on:

- the number of the *MonitoredItems* in a UA subscription;
- the size of the data in each *MonitoredItem* (each *MonitoredItem* was created on a OPC UA tag that was defined in server's address space as an Integer number);
- server status check interval, which has an important influence on how fast, after a server's failure, recovery actions at the client's side are started.

For each case, the total recovery duration was measured (t_R in (1)). This does not include the time taken for creating subscriptions and adding *MonitoredItems* to them, as it depends on the number of the subscribed tags and the data sizes.

Table 2. Measured times of failure detection.

Detection method	Publish interval	Failure detection time (ms)			
		Min.	Max.	Avg.	Dev.
service	1000	109	1000	500	200
Read	1000	550	2100	1000	240
keepAlive	1000	200	2300	800	650
service	100	109	140	120	18
Read	100	312	1450	110	300
keepAlive	100	577	2792	2050	900

Failure detection time

The preliminary test was performed to compare the server failure detection times t_D in three different situations (see previously): 1) direct server status check; 2) failed asynchronous read; and 3) timed out *keepAlive* message. Each of the scenarios was implemented and tested according to several parameters:

- the subscription's sampling interval;
- the subscription's publishing interval;
- the server status check interval (in 1) or asynchronous read frequency (in 2);
- inertial delay (time between the detection of failure and signalization that fact to the client; in an ideal case it is equal to 0 ms).

Test results of failure detection tests on three clients (Table 2) display big differences in measured times. It is the result of independent work of clients, with their own non-shared time parameters; the refresh requests from clients were not synchronized.

The test confirmed that the delay in detection of failure is neither dependent on the amount of *MonitoredItems* nor on the type of the redundancy. The main impact on the time required to detect the failure is from the parameters of the server and client, and the method used to detect the failure.

The 'Asynchronous read' method takes the most time and provides the least reproducible results; server status check is the fastest method but sometimes gives false warning even for time parameters that seem to be correct. When the time parameters of the connection are set incorrectly, e.g. when subscription sampling interval is more than 10 times less than subscription publishing, all the above methods detect false failure of the main server.

Table 2 presents the results of failure detection for three clients. The results are provided in the form of basic statistical measures of the sample, i.e. minimum, average and maximum time; standard deviation is also provided to give the information on samples distribution. Time parameters and server settings, for which the times have been measured, were respectively:

- subscription's sampling interval = 100 ms;
- asynchronous read frequency = 1000 ms;
- inertial delay = 100 ms;

- queue size at the server = 1;
- filter condition = OFF.

It should be noted that the distribution of the data is complex and hard to identify (see Polaków and Metzger, 2013, for an analysis of a similarly distributed data). There are multiple modes of the data distribution, which are explained by the complex nature of the investigated processes – passed messages are processed by complex multilayered protocol stacks of operating systems. Therefore, the measured times contain, amongst others, the execution times of applications, virtual machines, services and cores of operating systems involved. Because the experimentation is designed to provide a reader with an overall OPC UA timing performance, all the timing components had to be taken into account in the performance evaluation. However, the impact of the non-deterministic factors has to be minimized in order to make the experiments repeatable. In the described case, the minimization of the Windows 7 operating system influence on the communication timings was achieved by turning off all the non-crucial system services, automatic updates, system restore and hibernation capability. The running applications were limited only to those relevant to the experiments, including disabling of firewall and antivirus software.

It should be noted that in contrast to low-level hardware communication performance evaluations, in the described case the results heavily depend on the performance of not only the underlying hardware, but also of the operating systems and software, and when reused should be adjusted to the actual software/hardware environment.

Recovery actions time

As it was previously described, there are three distinct redundancy modes in OPC UA. The following experiment measures how the number of *MonitoredItems* influences the duration of recovery actions (i.e. switching from primary to secondary server) in each of the modes.

All the tests were performed in two hardware environments to determine the impact of the backup server's processing power on the results. Thus, there are two sets of results – for a backup server based on 1) a Pentium 8400 and 2) a Celeron 667 CPU. The results of the tests are shown in Figures 2–4.

As seen, the character of dependency of the recovery actions duration on the number of *MonitoredItems* is linear.

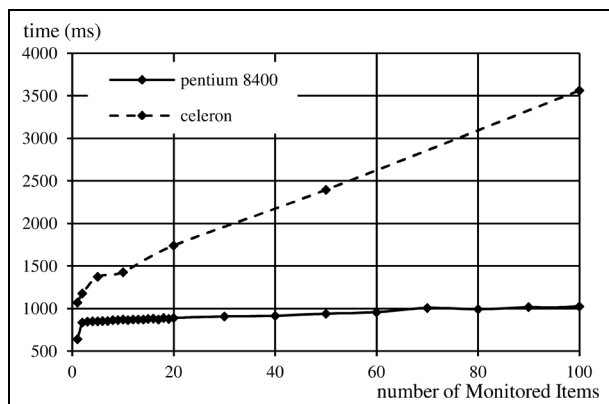


Figure 2. Duration of recovery actions depending on the number of *MonitoredItems* in COLD mode.

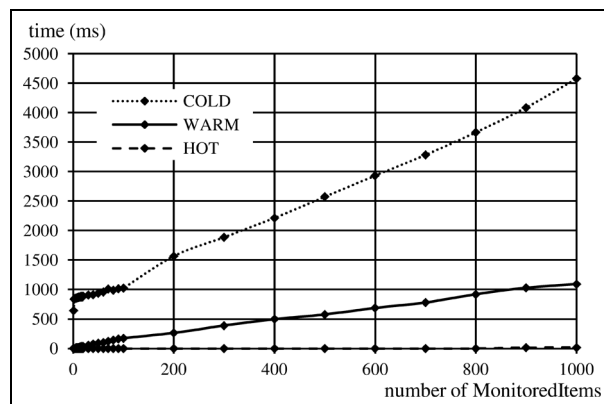


Figure 5. The comparison of recovery actions duration for all the three redundancy modes.

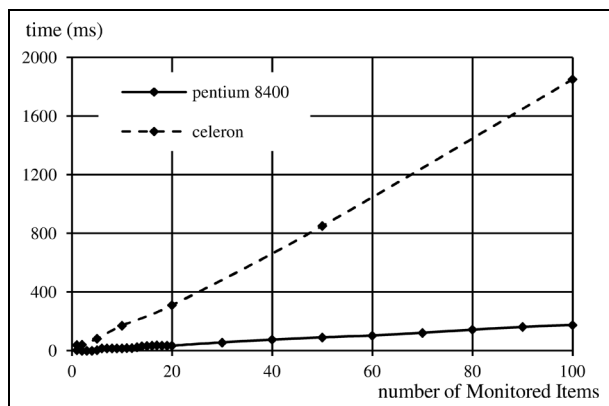


Figure 3. Duration of recovery actions depending on the number of *MonitoredItems* in WARM mode.

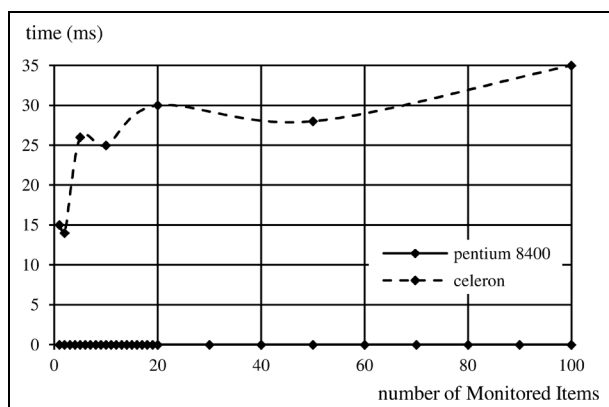


Figure 4. Duration of recovery actions depending on the number of *MonitoredItems* in HOT mode.

The odd shape of the graph for the HOT mode is explained by the precision of time measurement, which in case of the Celeron-based server was of the order of 10 ms. To compare

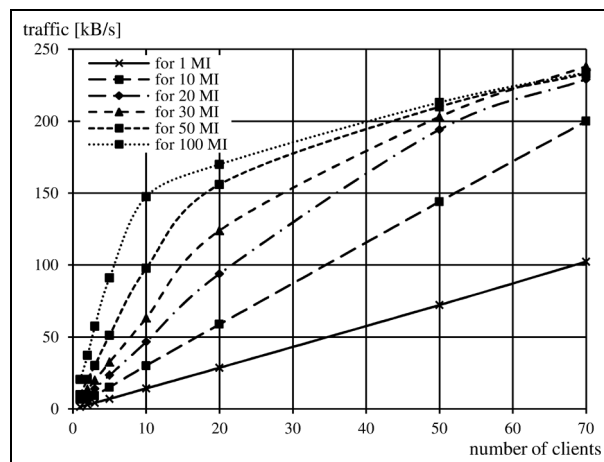


Figure 6. Measured network traffic from server to clients for different amount of clients and *MonitoredItems* (MI).

the results between the redundancy modes, the trend lines for the Pentium-based server (as the more representative for the current hardware) are presented in the same graph in the Figure 5 (for a wider range of number of *MonitoredItems* than in the previous figures).

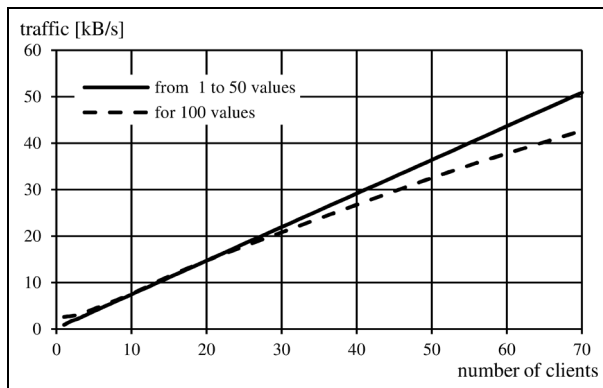
Other factors

The conducted experiments confirm that the failure detection time depends strongly on the time parameters of the subscription (Table 2), whereas the recovery actions duration depends primarily on the number of *MonitoredItems*.

Additional experiments were conducted in search of additional factors that could influence the measured times. Two parameters of the OPC UA communication in the experimental set-up, i.e. the generated network load and consumed CPU processing power, were measured in order to identify potential bottlenecks. Figure 6 shows the amount of generated traffic in the direction from servers to clients, depending on both the number of connected clients and the number of

Table 3. CPU load (%) of the backup server.

Number of clients connected	Mode	HOT			WARM			COLD		
		10	20	50	10	20	50	10	20	50
10	Connecting	7	36	72	15	38	80	0	0	0
10	Work	0	10	25	2	7	21	0	0	0
10	Switching	1	12	26	9	20	55	26	46	90
20	Connecting	22	70	73	22	40	90	0	0	0
20	Work	8	23	44	9	20	34	0	0	0
20	Switching	0	30	50	13	30	55	29	40	70
50	Connecting	25	98	100	30	90	100	0	0	0
50	Work	9	53	60	10	45	89	0	0	0
50	Switching	1	55	55	13	70	90	70	80	100

**Figure 7.** Measured network traffic from clients to servers.

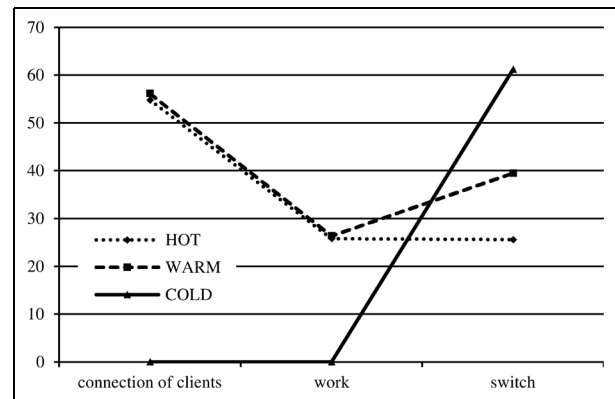
MonitoredItems. Figure 7 shows the amount of traffic in the opposite direction, and, as the values do not change much, the latter graph is simplified to illustrate the general character of the dependency.

During the tests, the network traffic measured at the server did not exceed 0.6 Mb/s (from all clients) and 2.8 Mb/s in the opposite direction. Total traffic was below 3.5 Mb/s on 100 Mb/s network segment. The measurements show that network load is not the critical factor for the relatively small amount of clients and *MonitoredItems*.

The results of the CPU load of the backup server are provided in a tabularized form (Table 3). The load was measured during three distinct phases of operation: connecting of the clients, normal work and switching after failure of the main server. Tests were performed for all the redundancy modes for various numbers of clients and *MonitoredItems*.

The results prove that the processing power of the machine hosting OPC UA server has significant influence on communication's quality and recovery actions time. However, this relation can depend on the specific implementation of the OPC UA, so if the results are to be reused they should be verified for the exact environment.

The data, after averaging, are presented graphically to illustrate the general trends of the CPU load (Figure 8). As seen, in the COLD mode there is virtually no CPU load at the

**Figure 8.** Comparison of the CPU power required in the three phases of redundant work.

first two phases of the redundant work. On the other hand, the difference between the HOT and WARM mode is seen only during the switching phase – the WARM mode requires additional subscription activation, which results in an additional CPU load during the last phase.

It also should be noted that the load generated by the OPC UA communication is not trivial; in the presented case servicing the *MonitoredItems* alone, without additional activation of the subscription, consumed over 25% of the CPU power. Therefore, when designing the hardware for the OPC UA communication, proper reserve of the performance power should be ensured.

Summary and future work

Performance evaluation provided in the paper gives an estimate of total recovery time for redundant OPC UA depending on the method of failure detection, time parameters for client and server, queue size, amount of *MonitoredItems* and redundancy mode. This knowledge is valuable in the industrial control systems in which it is required to monitor process parameters continuously, as illustrated by the examples provided previously. The provided results will be valuable for

those who will face the redundancy mode selection problem in their work.

To calculate total failure duration it is necessary to estimate the duration of failure detection t_D and the duration of recovery actions t_A separately (see (1)). As the performed experiments show, the t_D time depends on the used method of failure detection, whereas the t_A time depends on the number of *MonitoredItems*.

Estimation of the amount of lost data during the switching is not trivial, as the process consists of several steps. The exact number cannot be predicted, but the upper and lower bounds are determinable, by calculating the number of the data packages, which should be sent during the switching. For example, with the backup server based on Pentium 8400 CPU, with all time parameters set to 100 ms (and queue size = 1) in HOT mode for one *MonitoredItem*, the recovery action is taken instantly (time was lower than the accuracy of a single measurement). However, if the method used for the failure detection is based on *KeepAlive*, the time needed for detection is between 100 and 3000 ms, and during that time client will lose up to 30 *MonitoredItems* updates.

As another example, a case of COLD mode can be analysed. For the number of *MonitoredItems* of 50, the detection time (with the *KeepAlive* method) is again between 100 and 3000 ms. However, the recovery actions in this case take about 940 ms, which results in the total recovery time between 1040 and 3940 ms. It means that client will miss 10–39 *MonitoredItems* updates during the switching between the servers.

It is interesting, depending on the specific realization of the general bounds, that the client from the first example (COLD mode) could lose less information than the client from the second example (HOT mode). The conclusion is that the duration of the failure detection can be crucial and should always be taken into account.

The experiments were performed with the queue size limited to 1, but it is possible to increase the queue size on the server. However, this would result in a bigger delay in failure detection with the *KeepAlive* method. It should be considered to detect failure with multiple methods in such a case, e.g. *KeepAlive* and service status at the same time. This would make the failure detection independent from the queue size. This idea will be evaluated more closely and possibly experimentally verified in the future.

Another issue requiring attention is detection of false failures, which could happen when incorrect time parameters are set within the servers. Unfortunately, there is virtually no data available on the preferred values of the parameters and their interactions both in the manuals and in the literature. As this issue could lead to many problems in production environments, it should be noted that setting up redundancy is not trivial and should be performed only by qualified and well-prepared staff.

Moreover, some improvements would undoubtedly be advisable. In cases where no data loss is acceptable, NVP methodology can be used, but it requires further research, which is another task for the future.

Another interesting direction of further experimentation is the development of redundancy optimization agent, which would automatically choose the best mode and time

parameters. This would minimize the effort needed to provide a highly reliable OPC UA communication system and would be an attractive solution in heterogeneous environments, in which the deployment and configuration of redundant communication for each system node would be a repetitive and arduous task. The agent could be implemented in a way that allows switching between redundancy modes on runtime, according to continuously monitored changes of network load and/or data exchange parameters.

Conflict of interest

The authors declare that there is no conflict of interest.

Funding

This work was supported by the European Union from the European Social Fund and by the National Science Centre under grant No. 2012/05/B/ST7/00096.

References

- Alkaya A and Grimble MJ (2014) Non-linear minimum variance estimation for fault detection systems. *Transactions of the Institute of Measurement and Control*, (published online before print).
- Avizienis AA (1995) The methodology of N-version Programming. In: Lyu M (ed.) *Software Fault Tolerance*. New York: John Wiley & Sons.
- Baczyński D (2008) Koncepcja obliczeń technicznych i ekonomicznych dla potrzeb lokalnych rynków energii [Concept of technical and economic calculations for local energy markets]. *Rynek Energii* 4: 17–24.
- Cavalieri S and Chiacchio F (2013) Analysis of OPC UA performances. *Computer Standards & Interfaces* 36: 165–177.
- Chen I-R and Bastani FB (1994) Warm standby in hierarchically structured process-control programs. *IEEE Transactions on Software Engineering* 20(8): 658–663.
- Cupek R, Fojcik M and Sande O (2009) Object oriented vertical communication in distributed industrial systems. In: Kwiecień A, Gaj P and Stera P (eds) *Computer Networks*, CCIS 39. Berlin: Springer-Verlag, pp. 72–78.
- Du D, Fei M and Jia T (2013) Modelling and stability analysis of MIMO networked control systems with multi-channel random packet losses. *Transactions of the Institute of Measurement and Control* 35: 66–74.
- Felser M (2005) Real-time ethernet – industry prospective. *Proceedings of IEEE* 93(6): 1118–1129.
- Felser M (2008) Media redundancy for PROFINET IO. In: *IEEE International Workshop on Factory Communication Systems*, Dresden, pp. 325–330.
- Ferreira J, Almeida L, Fonseca JA, et al. (2006) Combining operational flexibility and dependability in FTT-CAN. *IEEE Transactions on Industrial Informatics* 2(2): 95–102.
- Folkert K, Fojcik M and Cupek R (2011) Efficiency of OPC UA communication in Java-based implementations. In: Kwiecień A, Gaj P and Stera P (eds) *Computer Networks*, CCIS 160. Berlin: Springer-Verlag, pp. 348–357.
- Frank PM, Ding SX and Marcu T (2000) Model-based fault diagnosis in technical processes. *Transactions of the Institute of Measurement and Control* 22: 57–101.
- Gessner D, Barranco M and Proenza J (2013) Design and verification of a media redundancy management driver for a CAN star topology. *IEEE Transactions on Industrial Informatics* 9(1): 237–245.

- Jammes F and Smit H (2005) Service-oriented paradigms in industrial automation. *IEEE Transactions on Industrial Informatics* 1(1): 62–70.
- Kasprzyczak L and Macha E (2015) Structures of the proportional-integral-derivative regulators for control of stress, strain and energy parameter at the electro-hydraulic fatigue test stand. *Journal of Vibration and Control* 21(1): 68–80.
- Lange J, Iwanitz F and Burke TJ (2010) *OPC – From Data Access to Unified Architecture*. Berlin: VDE Verlag, pp. 196–201.
- Mahnke W, Leitner SH and Damm M (2009) *OPC Unified Architecture*. Berlin: Springer-Verlag.
- Min D, Xiao Z, Sheng B, et al. (2014) Design and implementation of heterogeneous IOT gateway based on dynamic priority scheduling algorithm. *Transactions of the Institute of Measurement and Control* 36: 924–931.
- Moreira L, Spiess P, Guinard D, et al. (2008) SOCRADES: a web service based shop floor integration infrastructure. In: Floerkemeier C, Langheinrich M, Fleisch E, et al. (eds) *The Internet of Things*, LNCS 4952. Berlin: Springer-Verlag, pp. 50–67.
- OPC Foundation (2009) OPC Unified Architecture Specification Part 5: Information Model Release 1.01.
- OPC Foundation (2010) OPC UA Part 7 – Profiles 1.01 DRAFT Specification Part 7: Profiles Release 1.00.
- Patton RJ, Chen J and Nielsen SB (1995) Model-based methods for fault diagnosis: some guidelines. *Transactions of the Institute of Measurement and Control* 17: 73–83.
- Polaków G and Metzger M (2013) Performance evaluation of the parallel processing producer–distributor–consumer network architecture. *Computer Standards & Interfaces* 35(6): 596–604.
- Stebel K, Czczot J and Laszczyk P (2014) ‘General tuning procedure for the nonlinear balance-based adaptive controller. *International Journal of Control* 87(1): 76–89.
- Vyatkin V (2011) IEC 61499 as Enabler of distributed and intelligent automation: state-of-the-art review. *IEEE Transactions on Industrial Informatics* 7(4): 768–781.
- Yu J, Deng Z, Yu M, et al. (2013) Design of multiple controllers for networked control systems with delays and packet losses. *Transactions of the Institute of Measurement and Control* 35: 720–729.