ADI APPLIED DYNAMICS INTERNATIONAL

Industrial Real-Time Linux Servers

Introducing ADI's "rtx daemon" real-time Linux computing and data handling paradigm

Executive Summary

What is an Industrial Real-Time Linux Server?

An Industrial real-time Linux server is a computer server, low-cost or very high-end, with a real-time Linux operating system installed and providing a set of services for time-deterministic, frequency-based, computing and data handling. This real-time Linux server is used to connect your industrial capability, e.g. test facility, manufacturing line, electrical power system, into a Linux computing environment for the purpose of adding capability, e.g. predictive quality, anomaly detection, operational optimization, live data analysis, supervisory control, predictive maintenance.

What is the business value?

The deployment of real-time Linux servers is arguably one of the fastest and most cost-effective ways to move towards lights-out operation of your industrial capability. Most well-thought-out deployments see payback periods of less than 3 months. Key business value includes:

- Reduced quality related expenses
- Reduced operational costs

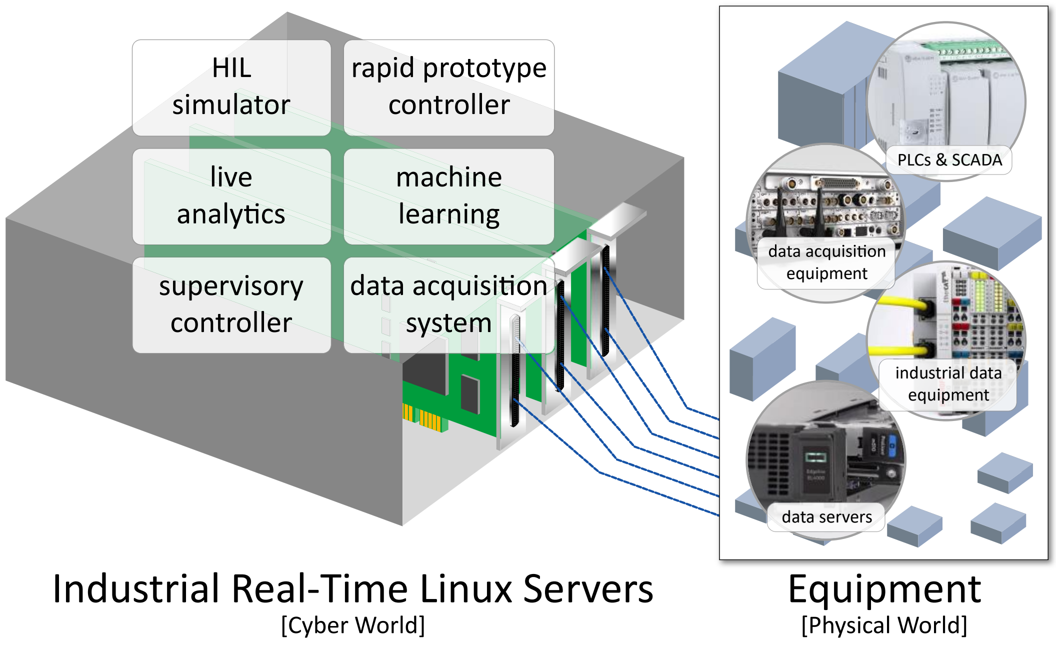


Figure 1 – Industrial Real-Time Linux Servers

The Challenge of Legacy Industrial Computing and Data Systems

There is a common source of pain across industry. You have a set of legacy rigs, systems, or capabilities with some level of real-time computing performance, based on internally developed software. Often your legacy software is based on commercial RTOS, e.g. VX-Works, QNX, etc., or an obsolete operating system, e.g. Pharlap, HPUX. These assets often begin to fall behind the state-of-the-art and may be anchored to obsolete computer equipment.

Your engineers and IT would like to move to a full Linux and real-time Linux deployment architecture. This offers the ability to connect your industrial systems to a flexible Linux computing environment, setup capabilities including monitoring, trigger, and analysis, do things like run Google's TensorFlow Machine Learning open source software, and connect into existing PLCs, computers, building management systems, and other equipment. The best part is such a Linux modernization will allow these systems to be brought into alignment with DFARS in support of your organization's cybersecurity hardening umbrella. The question is: How do you adopt a real-time Linux server modernization strategy without needing your IT and related engineering teams to become a full-blown software development organization? ADI's ADEPT Framework and rtx daemon provide a software platform enabling your team to design, implement, and manage real-time Linux servers with a small fraction of the non-recurring engineering costs.

Common Misconception: My Industrial Systems Modernization Efforts Don't Need Real-Time

If my industrial modernization project doesn't involve real-time control or real-time data acquisition, why would I need to consider real-time Linux computing?

Some industrial applications are obvious and require real-time, closed-loop control and/or real-time supervisory control. For other applications it's not always so obvious. Consider real-time computing as a tool for your industrial computing and data handling. You may have many functions, analysis, and data handling tasks that won't operate in a real-time, once-per-step type of computation. Many tasks and functions will be asynchronous and execute on command or in response to a condition or alert. A real-time Linux computing framework sets a high-resolution time reference for your computing environment. Accurate, microsecond resolution time measurement of events such as time of task initiation, time to complete task execution, and statistical variance of task behavior can be very useful for identifying and solving anomalous, difficult to find bugs in your operational systems. Real-time computation and analysis is very helpful and important for monitoring data throughput of mixed Industrial Ethernet and legacy industrial serial/network interfaces, e.g. Modbus, CANOpen, etc. Real-time computing and data handling may not be required for your application or it might turn out to be a very powerful tool, either way, open source real-time Linux costs the same as open source standard Linux. They are both free software.

Real-Time Linux versus Commercial RTOS

The commercial Real-Time Operating System (RTOS) market is big business with serious industry players, e.g. Intel. Ever since its release in 1991, Linux has been hailed by many thought leaders in the computer science community, as an existential threat to the RTOS market. In the early 2000's, Linux saw increased application for consumer embedded electronics devices. The commercial RTOS industry demonstrated the poor real-time performance exhibited by Linux and was able to draw a line in the market for where Linux would begin stealing market share and where commercial RTOS would continue to dominate and generate significant annual software licensing and support revenue.

In 2004, the Linux community released the "RealTime Preempt" patch, or RT Preempt. The Linux operating system, with its generic kernel, exhibits poor real-time performance as measured by real-time jitter. A key function of any real-time operating system is to allow frequency-based computing where a set of algorithms and/or data handling tasks are performed at every step, and executed at some frequency. Real-time jitter represents the time-based error associated with the operating system's ability to begin computation exactly at the beginning of every step. Today's commercial RTOS offerings typically perform with 3µs to 5µs of real-time jitter at three standard deviations. Traditional Linux, depending on the kernel configuration, could see real-time jitter as high as 500,000µs. RT Preempt dramatically improves the performance of open source Linux. Today, well-configured open source Linux with RT Preempt patch installed performs within the 3µs to 5µs jitter at three stddev range of commercial RTOS offerings. This means your engineers and IT folks no longer have to spend $10K to $20K per year for RTOS development tools, they can leverage the vast libraries of free open source software available for Linux, and you can benefit from far greater access to well-trained Linux development talent.

The rtx daemon – Linux Real-Time Executive

The "rtx daemon" is a set of Linux services which runs on ARM- and Intel-based computers and transforms them into full-featured real-time computers that are internally instrumented with performance, latency, jitter, and other precision time-based measurements for all aspects of the computer architecture. The rtx daemon (rtxd) software was developed, refined, and optimized by ADI over three decades and is now managed by the rtxd project. The goal of the rtxd project is to simplify the development and management of time-deterministic programming.

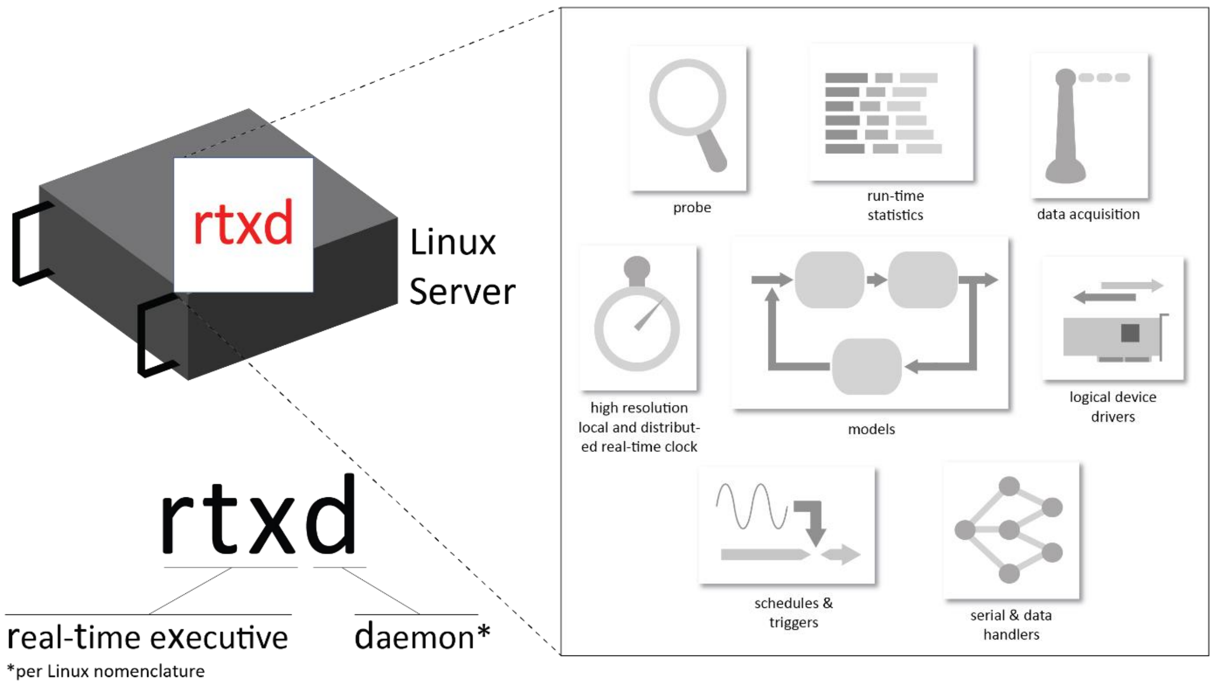
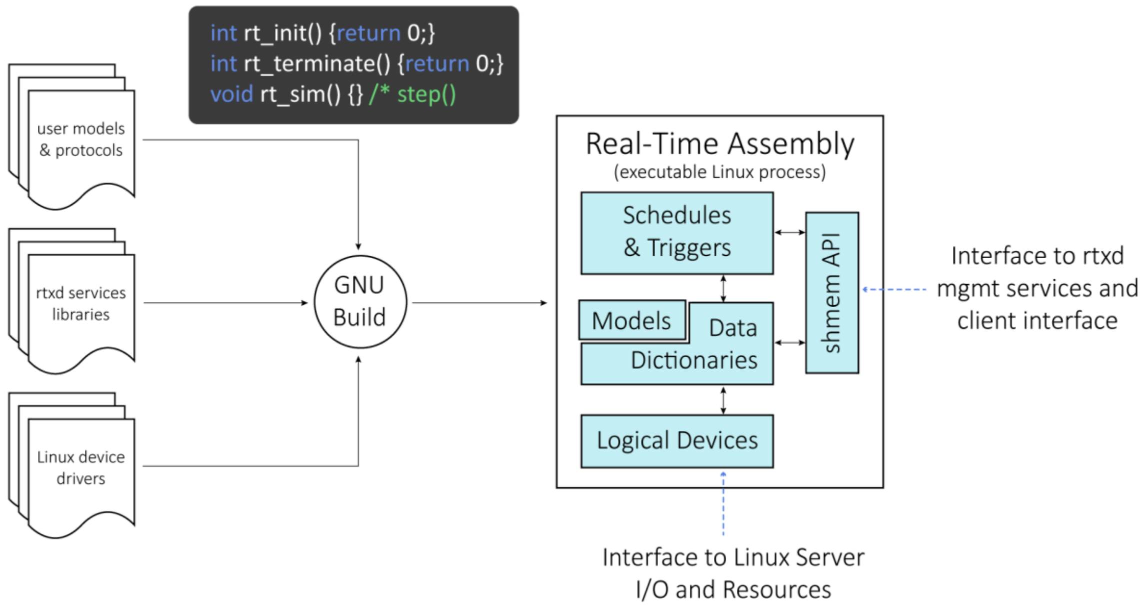


Figure 2 – rtxd Linux Real-Time Executive

By installing rtxd on your real-time Linux servers you get, right out of the box, a set of optimized real-time services with a multi-processor, multi-server distributed real-time computing and data handling architecture and a scalable client-server paradigm. The rtxd dramatically reduces the engineering and IT cost of modernizing your computing and data handling assets.

Real-Time Assemblies

A central element within rtxd's approach to scalable industrial real-time Linux servers is the "real-time assembly". Assemblies are used to package your computation and data handling tasks into one or more executable Linux processes which get uploaded to your real-time Linux server and are allocated to specific processor cores on a multi-core server (Intel or ARM) and allocated to specific Linux servers in a distributed, multi-server deployment. Figure 3 illustrates the components of a real-time assembly and build process.



```
int rt_init() {return 0;}
int rt_terminate() {return 0;}
void rt_sim() {} /* step()
```

Figure 3 – Real-Time Assemblies

Each assembly includes a C/C++ model that exposes the basic functions of initialize(), step(), and terminate(). ADI's ADEPT Framework client tools include functions for importing Simulink and FMI models into your real-time assembly. The simplest model provides only a set of variables in memory to read and write device driver calls in order to pass data in and out of data dictionary items. Data dictionaries are made up of a block of shared memory and an XML metadata specification for the dictionary, e.g. variable names, data types, dimensions, read/write access, data range, etc. Connections between data dictionary items and logical device ports provide the mechanism for performing real-time I/O device reads/writes.

The process of compiling a real-time assembly for ADEPT/rtxd links in user model code and protocol libraries, rtxd services libraries, and Linux device drivers to build an executable Linux process.

Assemblies are able to access data from interfaces such as computer boards, e.g. PXI, PCIe, and local I/O resources using Logical Devices, or LDVs. LDVs are a device driver layer that converts from procedural low-level Linux drivers to data "ports". Ports get connected to data dictionary items to read and write data from I/O channels, e.g. TCP/IP and UDP Ethernet, industrial Ethernet, e.g. Ethernet/IP, Powerlink, discrete inputs. LDVs can have features such as calibration, scaling, and protocol logic.

The rtx daemon services libraries also include dynamic, real-time scripting capability called "Schedules and Triggers" or S&T. S&T is implemented as a logic driven real-time perturbation engine acting off data dictionary items. S&T are commonly used to detect conditions based on a trigger expression and trigger an action, e.g. if TEMP>125 then send alert message and set several data dictionary value to TRUE.

Performance-Optimized Real-Time Computing – Method to the rtxd Madness

Real-time computing was once considered to be a bit of a black art. There are many reasons and opinions as to why it is viewed as unusual computing. A popular view suggests real-time computing tends to be foreign due to the way we typically learn to write software. We start with our "hello world" and we interact with the keyboard and the screen. Next we learn to allocate some memory for our program using a malloc(). Then we learn to read from and write to our hard disk drive for long-term data storage. We often are introduced to programming through the concept of a main() entry point and a procedural view of computing and task design.

In real-time computing, we switch to a frequency-based set of entry points, usually associated with a step() function in your code module. And there are a set of rules for implementing quality real-time code, including avoiding use of malloc(), avoiding disk reads and writes, and other best practices. In general, you try to avoid iterative numerical methods, e.g. solvers, integration algorithms, but sometimes they are unavoidable.

The rtxd software was designed and written with great effort invested in optimizing the services and architecture for real-time performance. A fundamental aspect of the rtxd high-performance real-time capability is associated with its data-centric "Data Dictionary" concept that makes use of shared memory structures and APIs for accessing data from all assemblies and connecting data from one assembly process to another with ultra-low-latency and computational overhead. This shared memory structure lends itself naturally to distributed, multi-server real-time framework deployments.

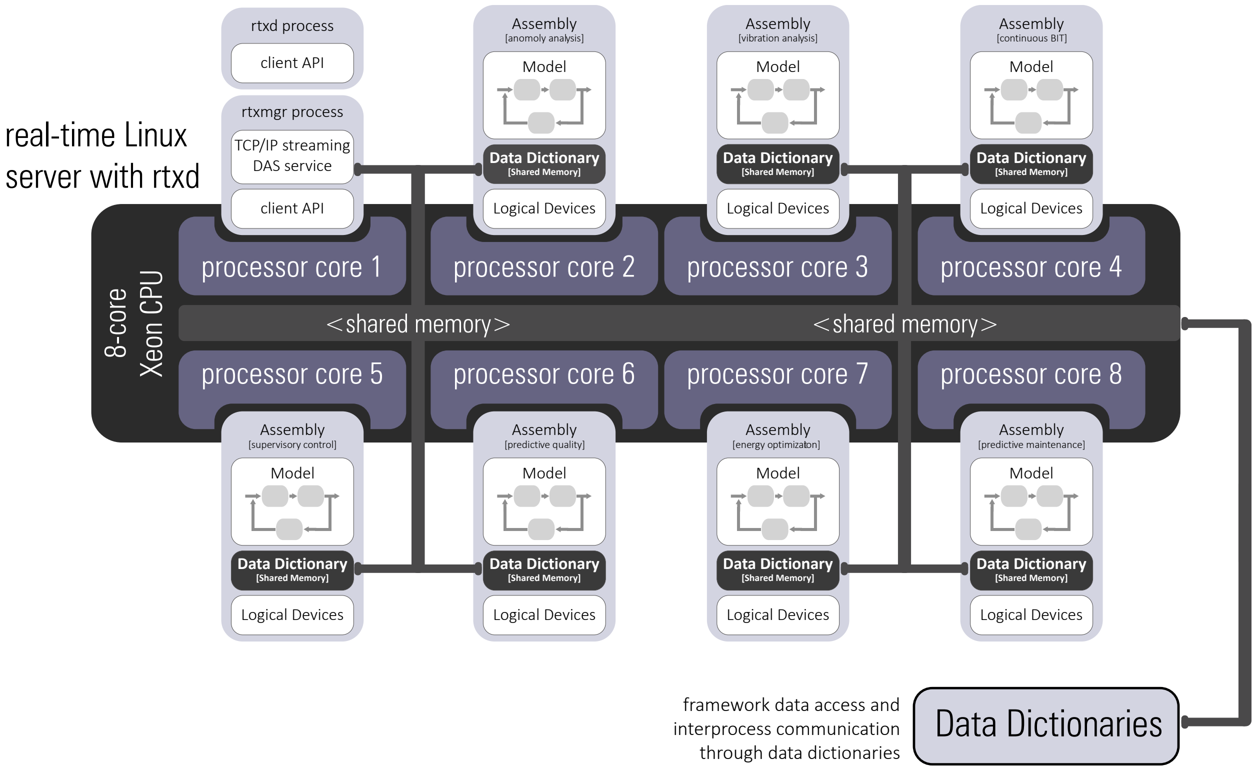

real-time Linux server with rtxd
rtxd process
client API
rtxmgr process
TCP/IP streaming DAS service
client API
Assembly [anomoly analysis]
Assembly [vibration analysis]
Assembly [continuous BIT]
Assembly [supervisory control]
Assembly [predictive quality]
Assembly [energy optimizaton]
Assembly [predictive maintenance]
Model
Data Dictionary [Shared Memory]
Logical Devices
8-core Xeon CPU
processor core 1
processor core 2
processor core 3
processor core 4
<shared memory>
<shared memory>
processor core 5
processor core 6
processor core 7
processor core 8
framework data access and interprocess communication through data dictionaries
Data Dictionaries

Figure 4 –Assemblies and Shared Memory Data Dictionaries

HISTORY NOTE: In the rtx daemon, the step() function API is provided with its rt_sim() function called at each step of real-time operation. The "sim" name was retained in the rtxd code to respect ADI's legacy providing real-time simulator deployments through the global aerospace and defense industry.

The decision to make use of compiled binary assembly executables as the basic container for computing and data handling tasks was made to get the very best real-time performance. Every computer has a fixed computational budget. As it says in its name, rtxd is a daemon, a set of services running in the background to implement a real-time computing framework. You get to choose how much of those services you want to make use of. You may write your Linux applications any way you like and you can stitch them into an assembly as much or as little as you like. You may just pass some basic operational and/or management data into an assembly's data dictionary to give you client access to your application. At the other extreme, you may use rtxd as the main entry points for all your industrial computing and data handling capability.

Real-Time Computing and Data Handling Frameworks

A real-time computing and data handling framework (real-time framework) is made up of one or more real-time Linux servers operating in coordination and with time synchronization. The real-time framework provides a flexible computational structure allowing these Linux servers to be interfaced with the real world, e.g. equipment, PLCs, sensors, actuators, data acquisition devices, etc. and operated as a time-synchronized cyberphysical system.

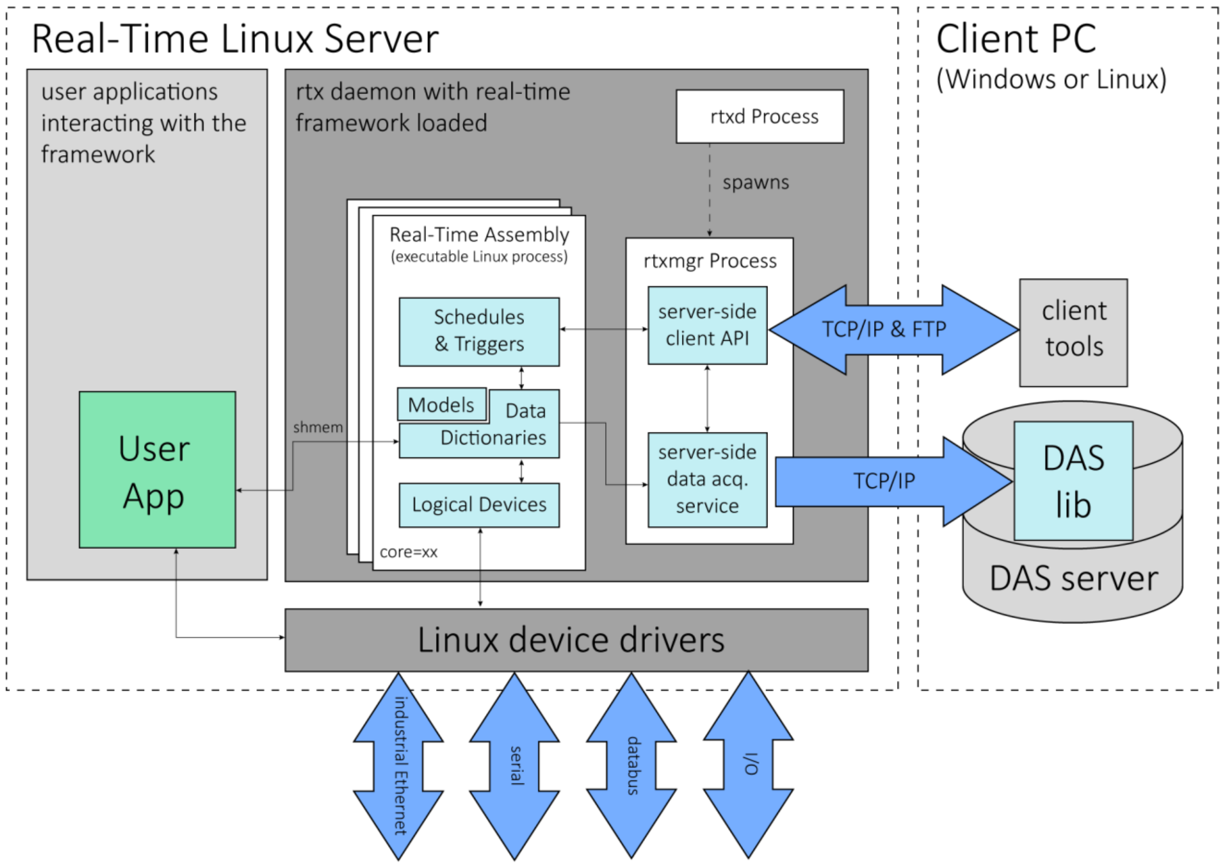


Figure 5 – Real-Time Assemblies Executing in an rtxd-Installed Linux Server

When a real-time Linux server boots, the rtxd process is spawned as part of the boot process. A real-time framework is loaded on one or more Linux servers by FTP'ing the set of assemblies to the participating servers along with configuration, data connections, and data dictionary XML files. After FTP'ing the framework files to the server, a framework load is commanded through the rtxd process API. This load command spawns the rtxmgr process which proceeds to execute and control each real-time assembly executable. Client tools interface with the framework through its TCP/IP API to get and put values, setup live streaming data, configure S&T, and interact through a wide set of functions.

The real power of industrial real-time Linux servers comes when you make use of multi-server distributed frameworks. The rtx daemon makes distributed computation and data handling a greatly simplified task by including both inter-process communication and inter-server communication. A primary rtxd server is assigned to act as synchronization and scheduling master for all Linux servers participating in the framework. Connections between data dictionary items are defined and if connected assemblies are allocated to different servers then the rtxd services automatically transfers data at each frame or selected frame multiple. The rtxd TCP/IP APIs allow multiple clients to connect and interact with the framework for management and operation

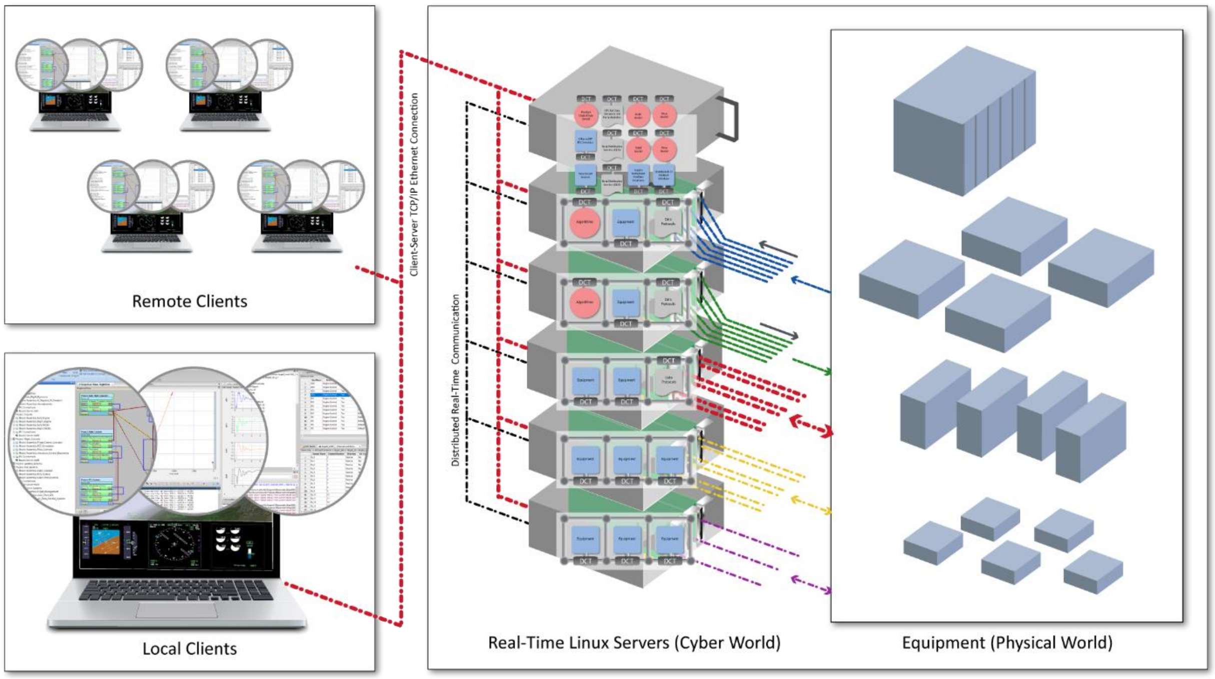

Remote Clients
Local Clients
Client-Server TCP/IP Ethernet Connection
Distributed Real-Time Communication
Real-Time Linux Servers (Cyber World)
Equipment (Physical World)

Figure 6 – Distributed Real-Time Framework Executing on Multiple Real-Time Linux Servers

Fast ROI with Phased Industrial Real-Time Linux Server Deployments

A nice feature of deploying industrial real-time Linux servers is the ability to connect into your industrial environment using a phased, step-by-step approach. Start by accessing the most valuable data you've always wanted to access and implement automated capability converting that data to value; typically quality-related and operational cost savings. You may start by connecting a small, low-cost Linux server into a single PLC or piece of equipment. Industrial real-time Linux deployments have the nice feature of connecting in with all the equipment you already have, rather than having to replace the equipment you have for something better. As your implementation of the Linux server matures you can connect into your building management systems, your data acquisition systems, and other capability to automate and work towards lights-out operation.

Case Study #1: Industrial Test Facility

Industrial real-time Linux servers are deployed in large industrial test facilities, such as jet engine test beds, to achieve a wide range of benefits including:

- Higher-accuracy test data - signal and time accuracy
- More comprehensive automated operation – Lights-Off Operation
- Reduced downtime and higher facility capacity utilization
- Reduced maintenance cost and asset cost

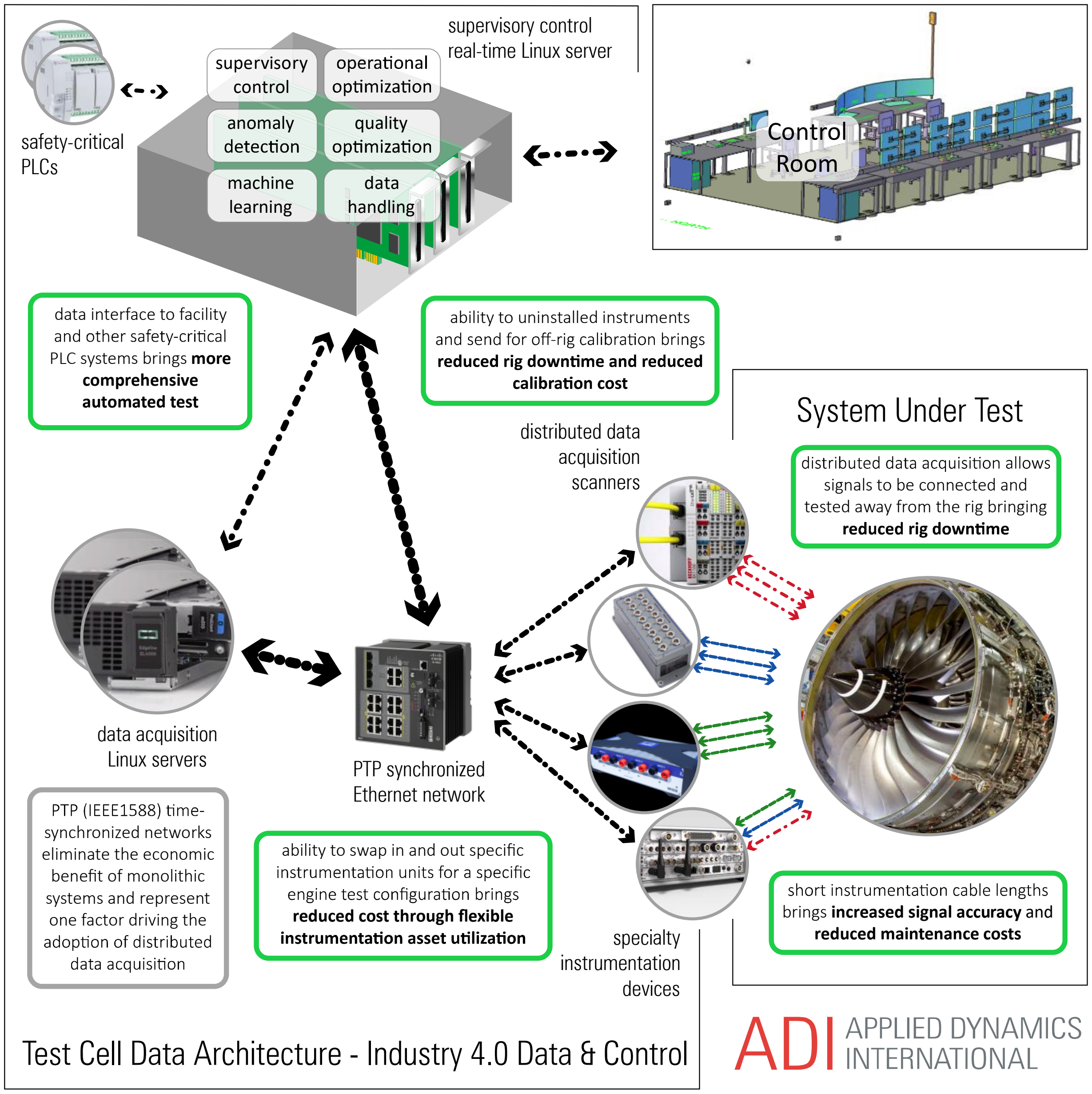

supervisory control real-time Linux server
supervisory control
operational optimization
anomaly detection
quality optimization
machine learning
data handling
safety-critical PLCs
Control Room
data interface to facility and other safety-critical PLC systems brings more comprehensive automated test
ability to uninstalled instruments and send for off-rig calibration brings reduced rig downtime and reduced calibration cost
System Under Test
distributed data acquisition scanners
distributed data acquisition allows signals to be connected and tested away from the rig bringing reduced rig downtime
data acquisition Linux servers
PTP synchronized Ethernet network
PTP (IEEE1588) time-synchronized networks eliminate the economic benefit of monolithic systems and represent one factor driving the adoption of distributed data acquisition
ability to swap in and out specific instrumentation units for a specific engine test configuration brings reduced cost through flexible instrumentation asset utilization
specialty instrumentation devices
short instrumentation cable lengths brings increased signal accuracy and reduced maintenance costs
Test Cell Data Architecture - Industry 4.0 Data & Control
ADI APPLIED DYNAMICS INTERNATIONAL

Figure 7 – Industry 4.0 Test Facility Data Architecture using Real-Time Linux Servers

Case Study #2: Fleet Monitoring and Control for 3D Printers

ADI has been working with the University of Michigan smart manufacturing research team to deploy real-time Linux servers to monitor and control fleets of 3D printers to detect anomalies, add predictive maintenance, predictive quality, and operational optimization.

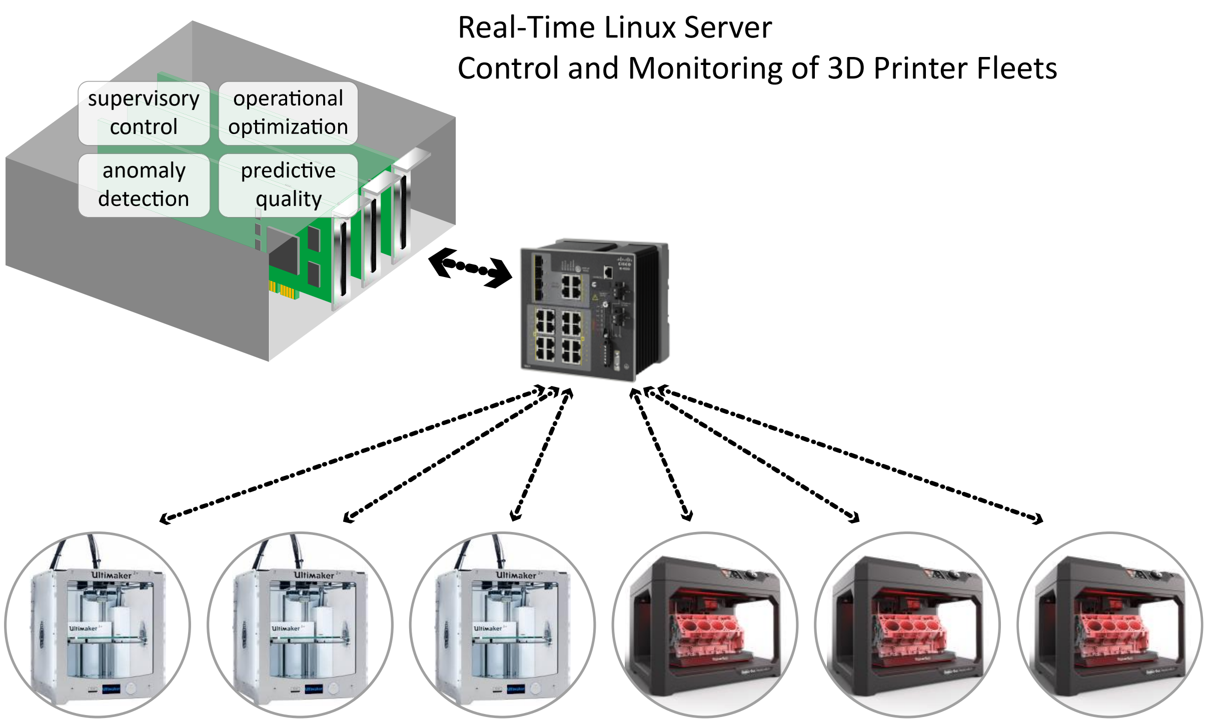


Figure 8 – Using Real-Time Linux Servers for Monitoring and Control of Fleets of 3D Printers

Case Study #3: Aircraft Cockpit Real-Time Integration, Verification, and Validation

A traditional use of real-time computing and data handling is for Hardware-in-the-Loop (HIL) testing to support aircraft systems integration, verification, and validation activities, including cockpit development. The transition from commercial RTOS based real-time systems to real-time Linux based systems for HIL testing can be seen across the global aerospace and defense industry.

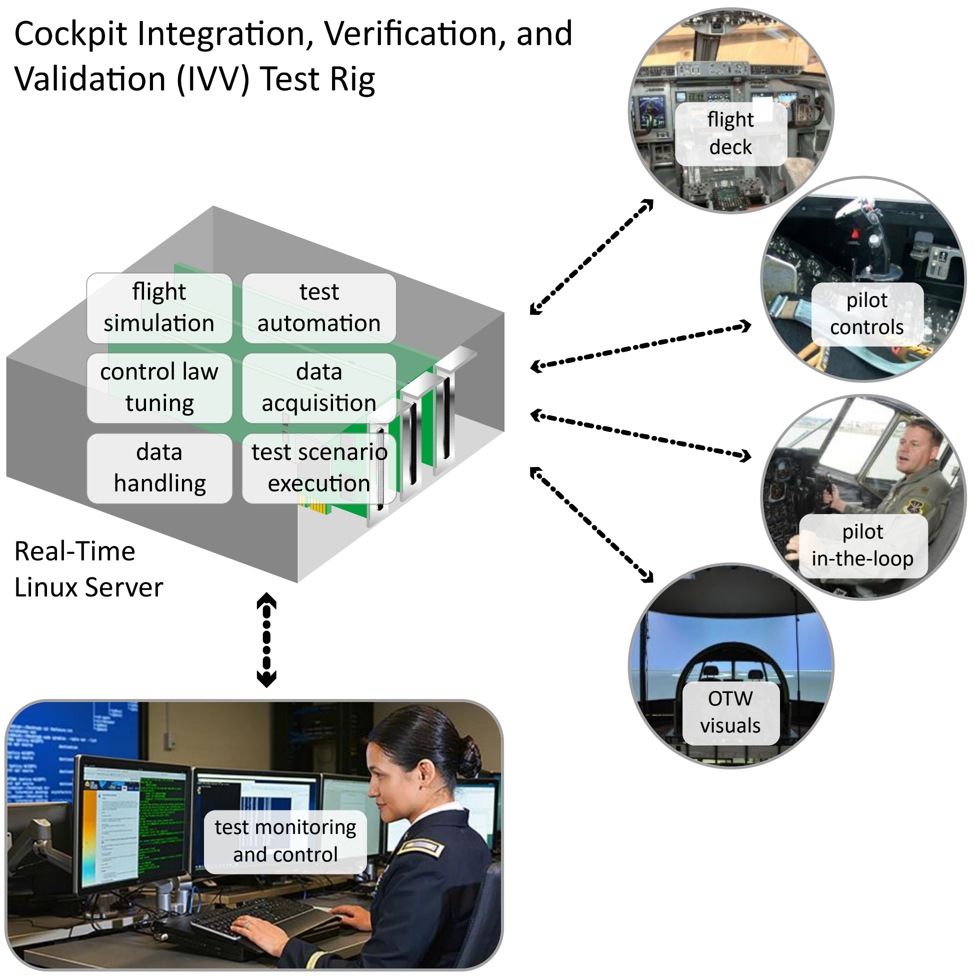


Figure 9 – Real-Time Linux Server Providing Core Computing and Data Handling for Cockpit Development and Verification

Go here to learn more about the ADEPT Framework: https://www.adi.com/products/adept-framework/

Go here to learn more about the rtx daemon: https://www.adi.com/resources/technical-library/