Navigating The Cloud: Enhancing Human-Robot Collaboration In Industry 4.0 Through Digital Twins And Cloud-Based Systems

Hayder Hasan Ali
University Mustansiriyah- Directorate Student Hostels
Asst. Prof. Dr. Khalil Mershad
Modern University For Business & Science (MUBS) / College Of Science

Abstract

Through This Detailed Examination, The Study Seeks To Provide Valuable Insights Into Optimizing Human-Robot Collaboration In Industrial Settings, Ensuring Both Security And Efficiency Are Upheld

**Keywords**: Digital Twins, Industry 4.0, HRC, Cloud

**I. Introduction**

Industry 4.0 Envisions Future Production Systems That Are Versatile, Extendable, And Readily Customizable. These Systems Provide Production That Is Not Dependent On Particular Items, Use Manufacturing Procedures That Are Suitable For Several Products, And Adapt Capacity Based On Current Demand Levels. Human-Robot Collaboration (HRC) Is Essential For Future Workflows To Achieve Flexible Manufacturing Processes. HRC Integrates Human Craftsmanship And Cognitive Abilities With The Accuracy And Power Of Robots. Novel Dangers Arise From This Distinctive Link And Are Governed By Specific Safety Regulations. Regrettably, There Is A Compromise Between Safety And Productivity, Perhaps Resulting In Reduced Production Due To Existing Requirements. The Robots' Mobility Significantly Affects Production. Accurate Control Of Robot Trajectories May Enhance Production While Upholding Safety Requirements [1].


This Article Analyzes Public And Private Cloud Platforms That Provide Function As A Service (Faas) Solutions And Compares Them To Conventional Methods. Function As A Service (Faas) Is A Robust
Alternative To Local Multi-Threading And May Greatly Lessen The Workload For Software Developers, Among Other Benefits. We Have Three Main Contributions. We Provide A Measuring Tool To Assess Different Parallel Processing Systems Based On Their Latency Attributes. We Will Analyze A Practical HRC Application That Involves Calculating Several Trajectories And A Key Performance Indicator (KPI). We Assess The Advantages And Disadvantages Of Various Techniques And Their Performance Attributes [3].

II. Related Work And Background
This Section Provides A Summary Of Previous Studies On The Control Of Industrial Robots And Describes Computational Virtualization Techniques Intended For Use. We Provide A Brief Summary Of A Highly Relevant Public And Private Cloud Platform. The Previous Studies We Present Explain What Has Been Achieved Regarding Control, And We Explain How It Is Measured In The Next Section.

A. Industry 4.0
Industrial Applications Are Crucial For The Internet Of Things (IoT). Cloud Robotics Is A Prominent Area Of Focus In Industrial Robotics Research, Since Recent Studies Have Shown The Advantages Of Linking Robots To A Centralized Processing Unit. A) Utilizing Advanced Computing Resources In A Centralized Cloud For Machine Learning (ML) Tasks; B) Reducing The Cost Per Robot By Transferring Functions To A Central Cloud For Sharing; C) Enhancing The Integration Of External Sensor Data And Enhancing Collaboration With Other Robots And Machinery; D) Enhancing Function Reliability By Running Multiple Instances As Hot Standby In The Cloud, Allowing For Immediate Takeover Of Operations From A Faulty Primary Function Without Interruption [4].

B. Novel Cloud Technologies

Effective Resource Allocation and Coordination Procedures are Essential for Arranging Virtual Control Functions within Specified Latency Limits in the Cloud for Applications requiring low latency. Internal operations of the Cloud Platform might impact the overall latency. In a prior research project, we conducted a thorough analysis of Amazon's Public Cloud Platform (AWS), specifically examining factors that contribute to delays and affect the performance of latency-sensitive applications in this setting. Choosing cloud services strategically during the design phase and ensuring the application can withstand delays of around 100 milliseconds allows for the feasibility of the cloud-native approach, along with its benefits. Edge and Fog computing are innovative technologies that improve standard cloud computing by placing computer resources in closer proximity to users and end devices. Several studies examine the potential uses, required technology, and coordination choices of edge computing. The key aspect of the Edge in Industry 4.0 applications is its capacity to quickly meet rigorous schedule requirements, even in the face of delays from internal operations [7].

C. Amazon’s AWS Platform

AWS, the leading public cloud provider, provides a range of choices to meet the processing requirements of Industry 4.0 applications. It provides three services that enhance computational performance and provide exceptional scalability. EC2 provides rapid virtual machine deployment on Amazon's infrastructure, offering a higher level of service. Lambda is a serverless platform that hides the underlying infrastructure, whereas Elastic Container Service (ECS) enables customers to install Docker containers on EC2 servers [8]...

Users mostly handle resource management. These choices differ significantly in cost and configuration. EC2 offers a variety of preconfigured packages called flavors, which include virtual CPU, memory capacity, and network speed. It allows the operation of many applications without stringent limitations. Users may choose virtual machines with a maximum of 128 virtual CPUs, enabling substantial parallelization at the instance level. ECS allows programs to operate across several platforms, but its instances are limited to accessing a certain range of resources. Containers may be allocated up to four virtual CPUs at most. Running many instances simultaneously may address this issue, but configuring them might be time-consuming.
Using Lambda Simplifies Code Scalability And Parallel Execution, But It Imposes Restrictions On The Kinds Of Code That Can Be Run And The Configuration Choices Users May Make. It Is Restricted To Certain Runtimes And Distributes Compute Resources According To Memory Configurations In An Arbitrary Manner. Lambda Relies On Event Sources Provided By AWS To Trigger A Function, While EC2 And ECS Instances May Be Accessed In Any Manner Chosen By The User. The Platform Generates New Instances Automatically When There Are None Available To Handle A Request, Up To A Set Maximum Limit. EC2 And ECS Charges Are Determined By The VM And Container Uptime, Whereas Lambda Pricing Is Based On Execution Time, Allotted Resources, And Request Count [10,11]...

D. Openwhisk

Openwhisk Is An Open-Source Function As A Service (Faas) Platform That Offers Several Deployment Options And A Distributed Serverless Cloud Solution. The Most Fundamental And Advisable Approach Is To Use The Kubernetes Container Management System, Which Accommodates Several Software Container Frameworks. Openwhisk Adheres To The Faas Programming Style, Allowing Developers To Create Actions, Which Are Functional Logic, In Many Supported Languages [12]. The Framework Conducts These Actions In Response To Certain Events Or Triggers, Regardless Of Their Size. Compared To A Virtual Machine Or A Complete Container, A Scale Unit Is A Software Feature That May Have A Smaller Footprint. The Platform Oversees The Whole Infrastructure, Including Both Physical And Virtual Servers, And Adjusts Docker Containers In Real-Time Based On Changes In The System's Requirements [13,14]..

Iii. Human-Robot Collaboration

Figure 1 Depicts A Use-Case Scenario Of Human-Robot Collaboration.

Figure 2 Displays Targeted Scenarios: Single VM (A), Synchronous (B), And Asynchronous Lambda Calls (C).

In The Future, Advanced Collaborative Robots Might Accurately Plan Their Paths To Avoid Accidents And Ensure Optimal Efficiency In Scenarios Where Humans And Robots Work Together [16]. Figure 1 Illustrates A Theoretical Future Situation Including The Use Of HRC. In A Complicated Manufacturing Setting Known As A Robotic Cell, A Sophisticated Robotic Program Controls Conveyor Belts For Transporting Workpieces, People, Or Cargo, A Robotic Arm, And Other Parts. We Are Interested In Managing The Robotic Arm [17]. The Primary Program Utilizes A Trajectory Controller To Guide The Arm To A Specified Location By Invoking A Trajectory Generator To Establish The Optimal Path. The Trajectory Of The Robot May Be Influenced By Various Limits And Goals Via Cooperation Between The Trajectory Executor And The Low-Level Controller. Reassessing The Current Approach May Be Required Considering The Cell's Interactive Behavior And Its Contacts With Human Persons. The Arm Should Navigate Around Obstacles Instead Of Stopping The Robot. The Complexity Of The
Cell Will Dictate The Computational Workload Required For These Computations. Multiple Contingency Routes Should Be Planned Ahead Of Time And Readily Available In Changing Situations [18,19].

Multiple Alternative Routes Are Computed From Different Starting Points Along The Current Trajectory Upon Detecting A Likely Collision. Figure 1 Displays Three Initial Locations Marked With POI Icons. A Variety Of Options With Distinct Characteristics Are Calculated From A Set Initial Location, And A Sophisticated Algorithm Is Used To Assess The Probability Of Item Collisions. Execute The Second Calculation For Each Segment Of The Authorized Path, Eliminating Any Duplications. Calculating Trajectories And Collisions Requires Several Instances Of These Processes To Operate Simultaneously[20]. The Quality Of Diversions And The Platform's Response Time Will Significantly Affect The Cell's Overall Performance, Particularly When Detours Are Easily Available. If A Detour Is Provided From The First Designated Location To The Second Indicated Area, The Robot Is Unable To Follow The Prescribed Path. If The Latency Requirements Are Met, Certain Components Of The Robot's Control System Might Be Operated In Cloud Computing Settings [22].

**Methodology:**

- Combination Of Cloud Services: Employing A Mix Of Public And Private Cloud Services To Explore Different Parallel Processing Techniques, Aiming To Optimize Performance While Mitigating Latency Issues.
- Practical Application: Demonstrating The Practical Application Of HRC In An Industrial Setting, Including The Deployment Of Advanced Collision Avoidance Systems.
Figure 3 The Three Main Contributions

Iv. Measurement Methodology
This Section Focuses On The Measuring Approach We Used To Analyze Various Deployment Choices For Comparable Applications And Program Patterns.

A. Simple Test Functions
Initially, We Use Hypothetical And Resource-Intensive Techniques For Preliminary Testing. Figure 2 Displays Three Distinct Setups That Were Developed To Evaluate The Effectiveness Of Parallelization With Function As A Service (Faas). We Examined A Scenario In Which Identical Operations Must Be Executed, But They Commence From Varying Initial Conditions. We Use Go Code To Run Resource-Intensive Tests And Gauge The Code's Execution Duration....
We Use A M5.24xlarge AWS EC2 Instance With 768 Gib Of RAM And 96 Vcpus With Basic Multi-Threading As Our Starting Point. Figure 3 Shows The Test Code Initiating Several Threads And Then Waiting For Each Thread To Complete Its Computations. Figure 2b Displays The Configuration For The Second Test Scenario, Using AWS Lambda For Concurrent Processing. Once The EC2 Instance Creates Threads, These Threads Then Invoke Lambda Functions Synchronously. While The Caller Threads Are In A Waiting State, The Lambda Functions Do The Specified Computations And Provide The Results To The EC2 Virtual Machine. We Can Calculate The Overall Execution Time In Both Scenarios By Subtracting...
The Start Time Of The First Thread From The Completion Time Of The Final Thread, As Seen In Figure 3. Figure 2c Depicts A Situation Where Lambda Functions Are Invoked Asynchronously By Threads Initiated By The EC2 Instance. The Outputs Of These Functions Are Stored In An In-Memory Redis Cache. Every Function Saves Its Result With A Distinct Key Provided By The EC2 Master. The EC2 Instance Consistently Retrieves The Output Of Each Lambda Function From The Cache. Execution Time In This Arrangement Is Defined As The Duration From The First Call To The First Lambda Function Until The EC2 Instance Can Access Each Provided Key. We Assess Situations On Our Openwhisk Platform By Using Actions For Both Synchronous And Asynchronous Lambda Calls, Rather Than Lambdas. We Implemented The Openwhisk Environment Using Kubernetes Following The Prescribed Approach. The AWS EC2 Instance Has A Total Of 96 Virtual Cpus.

B. Real Use-Case: Trajectory Calculation

Here, We Are Examining A Smaller Version Of The Use-Case Described In Section III. The Software Aims To Provide Alternative Paths To A Mobile Robot Arm To Prevent Collisions With Unforeseen Objects. Allocating The Program With The Appropriate Computing Capacity Will Enable It To Generate Several Trajectories For Chosen Places Ahead Of The Arm Reaching Them. The Elements Shown In Figure 5 Constitute The Custom Designed Software That Integrates Certain Functionalities. A Controller Supervises Jobs Near The Robot Arm And Interfaces With A C++ API To Get Longer Trajectories. Figure 1 Illustrates The C++ API, Which Is A Locally-Operated Multi-Module System. It Is Responsible For Sending Requests To The Cloud For Assistance And Receiving Responses From Several Local Controllers To Support Many Robots. We Established An EC2 Virtual Machine Instance In The Cloud To Execute Simultaneous Lambda Function Calls Following The Steps Shown In Figure 2b. Lambda Functions Are Used To Calculate Object Collisions, Unlike Other Functions Which Are Executed Locally On The Virtual Machine. The Important Performance Metric Is The Quantity Of Beneficial Trajectories Computed By Our Program Within A Certain Timeframe.
V. Experimental Evaluation
This Section Presents The Primary Findings Derived From A Thorough Performance Study Conducted Using Our Measuring Methodology.

A. Results With Simple Test Functions
We Used The Conventional Go Test Function To Do Many Measurements. The Tests Are Created To Compare And Evaluate The Efficiency Of Several Systems Running Simultaneously. We Specifically Assess The Performance Improvement Of Function As A Service (Faas) Systems With Either Unlimited Or Restricted Virtual Cpus Compared To The Additional Workload Caused By Background Activities. Various Test Functions With Varying Computing Complexities Are Assessed To Determine Their Impact.
On Overall Performance. Table I shows the performance timings of certain synthetic functions evaluated on a standard CPU core. We contend that a crucial factor affecting total performance is the quantity of parallel workers, such as threads, Lambda functions, or actions. We used 10, 100, 200, 500, and 1000 parallel workers sequentially for process analysis. Figure 6a displays the average execution times of different workers on all platforms. The function's complexity directly impacts the execution time on both AWS and Openwhisk. Due to AWS providing a limitless amount of cores, the desired outcome is attained. Openwhisk maintains its speed on a virtual machine with 96 virtual cpus even while installing 1000 activities. This circumstance happens when actions are placed in a queue before being scheduled. Even with 10,000 concurrent activities allowed in the Openwhisk configuration, operations may still experience significant wait time. The multi-threaded version of the test software shows decreased performance when using all 96 available cores, as anticipated...

Secondly, Figure 6b displays the total execution time for several situations involving one thousand workers. Two reference measurements are provided for multi-threaded processes: one using a big virtual machine (VM) with 96 cores and the other using a smaller VM with 8 cores. An important distinction is expected. AWS Lambda calls, whether synchronous or asynchronous, provide outstanding performance for each specified scenario. When dealing with huge functions, AWS solutions are more effective than a multi-threaded program running on a large virtual machine. The medium-sized virtual machine (VM) routinely exhibits worse performance compared to AWS. The workers' complexity greatly influences the outcomes of Openwhisk. The platform overhead and queueing delay are caused by several factors, such as internal scheduling and Docker in Docker. With 1000 workers, synchronous and asynchronous modes provide comparable performance overall...

Figure 7 illustrates the performance parameters based on the number of staff. The findings demonstrate the comparison between medium-sized workers using synchronous (Fig. 7a) and asynchronous (Fig. 7b) calls, using the multi-threaded operation as a reference point. Openwhisk works in two methods, with AWS synchronous calls being somewhat faster than the asynchronous technique. Openwhisk demonstrates superior performance compared to AWS with a smaller number of workers (10), suggesting that AWS may have a greater platform
Overhead. The Total Performance Decreases As The Number Of Workers Equals Or Exceeds The Number Of Underlying Cores In Openwhisk, As Anticipated. Openwhisk's Scalability Remains Stable Even As The Number Of Workers Rises, Unlike AWS. During Our AWS Trials, You Have The Freedom To Use Unlimited Cores And Lambda Functions Simultaneously. The Findings Validate That Openwhisk Is Effective As A Function As A Service (FaaS) Platform In Private Edge Cloud Deployments...

**B. Results With Trajectory Calculation**

We Incorporated The Pertinent Sections Of The HRC Use-Case From Section III Into A C++ Program Designed For Use With Universal Robots' UR5 Industrial Robot Arm. We Adhered To The Technique Outline In Figure 5 Of Section IV-B Throughout The Studies. We Assessed Several Methods For Simultaneous Object Collision Detection For Each Segment Of The Trajectory Being Studied. This Key Aspect Directly Impacts The Number Of Beneficial Alternative Paths That May Be Shown To The Robot Within A Certain Timeframe. Figures 8a And 8b Show The Robotic Arm's Movement In Cartesian Space While Operating On Our Large Virtual Machine Hosted On AWS, For Single-Threaded And Multi-Threaded Implementations, Respectively. The Top View Of Our GUI Displays Red Curves In The Graphs Representing The X-Y Movement Of The Arm's Termination. This Deployment Might Provide More Efficient Alternative Routes With Fewer Accidental Deviations. Our Solution Transitioned To The AWS Lambda Framework, But Saw A Significant Decrease In Performance Because Of The Use Of The Older C++ Runtime (V0.1.0). Substantial Delays, As Seen In Figure 8c, Led To Operational Failure...

Table I Displays The Complexity Of The Functions In Reference Measures.

<table>
<thead>
<tr>
<th>Label</th>
<th>Execution Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Xsmall</td>
<td>~5 Msec</td>
</tr>
<tr>
<td>Small</td>
<td>~10 Msec</td>
</tr>
<tr>
<td>Medium</td>
<td>~20 Msec</td>
</tr>
<tr>
<td>Large</td>
<td>~50 Msec</td>
</tr>
<tr>
<td>Xlarge</td>
<td>~100 Msec</td>
</tr>
<tr>
<td>Xxlarge</td>
<td>~200 Msec</td>
</tr>
</tbody>
</table>
(A) Mean Execution Duration Of Workers With Varying Levels Of Complexity

(B) Total Execution Time Of Jobs With 1000 Concurrent Workers

Figure 6 Displays The Execution Times Per Worker And Total On Various Platforms.

(A) Synchronous AWS Lambda Invocations, Synchronous Openwhisk Functions, Multi-Threading On A Virtual Machine With 96 Virtual Cpus.
(B) Asynchronous AWS Lambda Calls, Asynchronous Openwhisk Actions, Multi-Threading On A VM With 96 Vcpus (As A Reference). Figure 7 Displays The Overall Performance Across Several Platforms When Using Medium-Sized Personnel.

(A) Single-Threaded Implementation. (B) Implementation Using Multiple Threads

(C) Lambda Execution Times Histogram
Figure 8 Displays The Paths Of The Robot Arm In Cartesian Space In Panels (A) And (B), Together With A Histogram Of A C++ Lambda Function On AWS In Panel (C).

Vi. Conclusion
Simultaneously To Provide Timely Control Choices For Avoiding Collisions. We Analyzed The Duration Required To Complete The Task For These Computations In Three Unique Implementation Situations. Openwhisk Operates On A Potentially Private Cloud, AWS Lambda Functions In A Public Cloud, And A Conventional Multi-Threaded Application...


In The Future, We Want To Build The Application Using Go To Take Advantage Of Improved Support In Faas Platforms And Reduce End-To-End Latency By Using Private Edge Clouds. Aligning The Amount Of Parallelization With The Number Of Computing Cores Is Essential For Maximizing Openwhisk's Performance On The Edge..
References


التنقل في السحايا: تعزيز التعاون بين الإنسان والروبوت في الصناعة 4.0 من خلال التوائم الرقمية والأنظمة المستندة إلى السحايا

خبير حسن علي
الجامعة المستنصرية - قسم شؤون القدام الداخلية
أ.م.د. خليل مرشد
الجامعة الحديثة للإدارة والعلوم (MUBS)

مستشفى البحث:
تهدف الصناعة 4.0 إلى إنشاء أنظمة بيئية صناعية متقدمة تتطلب تعاونًا بين البشر والروبوتات، وهو ما يسمى التعاون بين الإنسان والروبوت (HRC). وفي هذا الإطار، يستفيد مركز HRC المحتمل، وبالتالي حساب مسارات بديلاً لتجنب أي اتصال مباشر. علاوة على ذلك، يظهر مفهوم التوائم الرقمية كحل مبتكر، حيث يوفر منصة لتقديم عواقب استراتيجيات التحكم المختلفة في بيئة محاكاة رقمية في وقت واحد. في حين أن البنية التحتية للساحبة الحسابية المتقدمة يمكن أن توفر القدرة الحسابية المطلوبة، فإنها قد تقدم في الوقت نفسه تحديات مثل زيادة زمن الوصول والتغلب في الاتصالات، المعرفة باسم الارتعاش، والتي يمكن أن تعقيد النظام.

تشير التدابير الساحبة ل sistem ألات تحفيز عملاء العمل من مظلي الأنظمة، كما يثير تساؤلات حول تكاملها مع منهجيات التوائم الرقمي ومزودة الأنظمة الآلية في مواجهة التحديات المحددة الناتج عن الخدمات المستندة إلى السحايا، ويعد هذا البحث إلى التغلب على هذه التحديات من خلال دمج مزيج من الخدمات السحابية العامة والخاصة، والتي تتميز بقدرات المعايير المتوازنة الفريدة. مساهمات هذه الدراسة هي ثلاثة أضعاف. أولاً، يقدم طريقة جديدة مصممة لقياس مدى فعالية الاستراتيجيات المتكونة من خلال التركيز على تأثيرات المكون المرتبطة بها. ثانياً، فهو يحدد تطبيقًا ملموسًا لمجلس حقوق الإنسان، ومسهل الوضوء على قواعد العملية وقابلية تطبيقه في العالم الحقيقي. وأخيرًا، فهو يحدد مقياس أداء حاسم يهدف إلى تقييم كفاءة هذه الأنظمة بدقة. ومن خلال الخوض في هذه الجوانب، يهدف البحث إلى إجرء تحليل شامل للفوائد والضعف الكامنة في المنهجيات التكنولوجية المختلفة وتأثيراتها المرتبطة على أداء أطقم HRC ضمن نطاق الصناعة 4.0. ومن خلال هذا البحث التفصيلي، سيسعى الدراسة إلى تقديم رؤية قوية حول تحديد التعاون بين الإنسان والروبوت في البيئات الصناعية، مما يضمن الحفاظ على الأمن والكفاءة.

الكلمات الرئيسية: التوائم الرقمية, الصناعة 4.0, HRC, السحايا.