1. **Introduction**

Pravega is an open-source storage system implemented and led by Dell Technologies. It uses Streams as a first-class primitive which are based on the append-only log data structure. They are flexible and have good performance [1]. By implementing clients for Pravega in multiple languages, its use can expand to a variety of applications. It currently has clients in Java, Rust, and Python.

The goal of this project is to take the existing Pravega API client that is written in Rust and to create a wrapper for it in C#. The implementation should be seamless and behave the same way as any other C# library. By doing this, the .NET Framework, one of the most popular in the world, will be able to be used with Pravega.

1. **Background and Related Work**

Dell has an open-source project called PravegaAPI which delivers a distributed storage system for elastic data streams. The Pravega client is written in Rust, however they want to be able to access it in C# for broader access. Our team needs to be able to wrap all the functionality of the PravegaAPI in C#.

Tools have already been created to help efficiently wrap the code. There is a mechanism called foreign function interfaces (FFI) that allows you to call functions in one language written in another. When using FFI with Rust, it converts the code to C. You then convert the C code into C# for this development. There is also a framework called safer\_ffi [2] which allows the Rust code to not be filled with unsafe code blocks since we will be writing to unsafe C code. Another helpful resource is a blogpost [3] talking about wrapping Rust in C#. The blogpost is especially helpful in explaining how they handle memory allocation. This is extremely important because Rust has its own ownership system and C# has a garbage collector. By just using FFI without implementing a system to handle memory, you would run into problems. Instead of manually wrapping each function in Rust, it is possible to automatically wrap them. With a tool such as interoptopus [4], it attempts to make every binding look idiomatic with a large variety of languages supported. While this tool can save time it won’t be able to wrap everything perfectly and correct.

To be able to effectively complete this project for Dell, all members on the team will have to familiarize themselves with these technologies. The first is the Rust language, each member should understand the basics, especially the barrowing system. Two great resources for learning Rust is the Rust Documentation Introduction [5] and TourOfRust [6], for a more hands-on approach. Each teammate should also research FFI, specifically for wrapping Rust. One more language all members must know is C# and .NET. Microsoft provides some interactive tutorials [7] if you need an introduction. Lastly a good understanding of the C language is important. Everyone should be able to read C code and have a clear understanding of what it is doing.

1. **Project Overview**

The main problem this project is trying to solve focuses around the Pravega API. While the API supports many languages such as NodeJS, Rust, and Python for examples, it doesn’t work in the .NET framework and C#. What this means is that developers wanting to use the Pravega API for data streaming cannot use it on their .NET applications without having to manually wrap the API themselves, which is incredibly tedious for a user. This also means that the scope of problems that Pravega can be applied to is limited as it is bound by the languages it supports and cannot support .NET applications.

The objective of this project is to create a wrapper for Pravega such that it can be interpreted by C# in a .NET framework. Doing so will allow development using Pravega by a larger pool of businesses and skilled .NET developers who can use it for critical applications. Modern businesses require efficient and secure data transportation and Pravega can help more businesses accomplish this once this project is complete. By the end of the project’s timeline, the goal is to have a wrapper or some method of processing Pravega functions such that it can be interpreted in C#. This includes all Pravega Modules, which are byte, client\_factory, error, event, index, and sync [8].

The first step in this process will be researching Rust, C#, and their similarities and differences from one another, such as how C# uses a garbage collection method for memory disposal while Rust doesn’t. Finding similarities and differences between the interfaces will allow us to plan what can be done easily between the two languages and what needs to be accounted for in development. In addition, it is important to choose a framework designed for wrapping Rust functions into C#. Testing needs to be done on an individual function basis in order to see what framework wraps Rust functions the best. After we have decided on our framework and method of converting functions from Rust and wrapping them into C#, we will begin working on each module, wrapping more basic modules before tackling more complex modules. The “byte” module is the first target.

Our major milestones in this project will be developing a method of wrapping each function that can be applied to most other functions in Rust as summarized by the previous section as well as having all modules converted from Rust to C#. By the end of the fall semester at Washington State University, roughly late November, we plan on having the first milestone at a minimum complete. This includes having researched and tested frameworks that can mediate between C# and Rust as smaller milestones completed before the wrapping method is developed. The next milestone will be pertaining to each module as each module is another step closer to the final goal. After the first module byte, we will move onto client\_factory, then index, and so on. The project ends once the final module is complete and the public Pravega API wrapper is stored onto GitHub, which our team plans on having created by the end of spring semester, roughly late April. This is to act like a C# library and to appear as such for the user. A user not familiar with Pravega should be able to use this with little issue and a transition for an experiences Pravega user should be seamless. If time allows, discussion about implementing simple cyber security principles into the library is planned, but not urgent as Rust is an extremely secure language.

1. **Client and Stakeholder Identification and Preferences**

The Client for this project is Dell Technologies. They will be guiding and assisting us with the creation of the C# wrapper. They are also our primary stakeholder, as our project will enhance their Pravega system with the ability to be used in the .NET framework, further increasing its usefulness.

Our stakeholders also include the students working on the project. By successfully completing it, not only will we fulfill the requirements to graduate, but we will gain experience working with Dell, a global technology company. Potential future users are also stakeholders in the project. They could use Pravega in future application to handle data storage and management.

The largest preference for all of our stakeholders is that the wrapper is seamlessly implemented in C#. It should be able to be called and used like any other library in C#. This will ensure its maximum usability and will increase the chances that it will be used by developers.

**Glossary**

* API-Application Programming Interface

**References**

[1] “Pravega concepts¶,” *Concepts - Exploring Pravega*. [Online]. Available: https://cncf.pravega.io/docs/v0.11.0/pravega-concepts/. (accessed Sep. 20, 2022).

[2] “What is safer\_ffi?,” *GitHub*, Sep. 19, 2022. <https://github.com/getditto/safer_ffi>

(accessed Sep. 20, 2022).

[3] “How we integrate Rust with C#,” *Structured Blog*, Sep. 18, 2018.

https://blog.datalust.co/rust-at-datalust-how-we-integrate-rust-with-csharp/ (accessed

Sep. 20, 2022).

[4] R. Biedert, “ralfbiedert/interoptopus,” *GitHub*, Sep. 18, 2022.

<https://github.com/ralfbiedert/interoptopus>

[5] “Introduction - Rust By Example,” *doc.rust-lang.org*

<https://doc.rust-lang.org/beta/rust-by->example/index.html (accessed Sep. 20, 2022).

[6] “Tour of Rust - Let’s go on an adventure!,” *tourofrust.com*. <https://tourofrust.com/> (accessed Sep. 20, 2022).

[7] BillWagner, “Introduction to C# - interactive tutorials,” *learn.microsoft.com*. https://learn.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/tutorials/ (accessed Sep. 20, 2022)

[8] “Crate Pravega Rust Documentation,” https://docs.rs/pravega-client/0.3.1/pravega\_client/ (accessed Sep. 16, 2022).