Informe de Modificaciones Realizadas en el Repositorio ARBITRAGEXPLUS-II

Este informe detalla las modificaciones realizadas en el repositorio `ARBITRAGEXPLUS-II` para integrar la lógica de cálculo diferencial, la adquisición de datos en tiempo real, la validación de direcciones y otras mejoras, con el objetivo de transformar el bot MEV existente en una herramienta de arbitraje de alto rendimiento.

Resumen de Cambios

Se han realizado cambios significativos en la estructura del proyecto, añadiendo nuevos módulos y modificando los existentes para incorporar las funcionalidades avanzadas. Los principales archivos afectados son:

- `rust-mev-engine/Cargo.lock` : Actualización de dependencias.
- `rust-mev-engine/Cargo.toml` : Adición de nuevas dependencias (`tracing` , `tracing-subscriber` , `tracing-test`).
- `rust-mev-engine/src/database.rs` : Modificación de las estructuras `Opportunity` y `Execution` para incluir `kit_de_armado` .
- `rust-mev-engine/src/executor.rs` : Refactorización de la lógica de ejecución, integración de `KitDeArmado` , simulación pre-trade, gestión de gas dinámica y monitoreo de transacciones.
- `rust-mev-engine/src/mev_scanner.rs` : Adaptación para utilizar el nuevo motor matemático, `DataFetcher` y `AddressValidator` .
- `rust-mev-engine/src/monitoring.rs` : Adición de métricas de riesgo y contadores para transacciones revertidas.
- `rust-mev-engine/src/types.rs` : Creación de nuevas estructuras de datos (`KitDeArmado` , `PoolReserves` , `DexFees` , `GasCostEstimator`).
- `rust-mev-engine/src/data_fetcher.rs` : Nuevo módulo para la adquisición de datos en tiempo real.
- `rust-mev-engine/src/math_engine.rs` : Nuevo módulo para la implementación del cálculo diferencial.
- `rust-mev-engine/src/address_validator.rs` : Nuevo módulo para la validación de direcciones y contratos.
- `rust-mev-engine/src/lib.rs` : Actualización de las declaraciones de módulos.
- `rust-mev-engine/tests/test_mev_scanner.rs` : Creación de pruebas unitarias para el `mev_scanner` .

Detalles de las Modificaciones

`rust-mev-engine/Cargo.lock` y `rust-mev-engine/Cargo.toml`

Se han actualizado y añadido varias dependencias para soportar las nuevas funcionalidades, incluyendo librerías para `tracing` (para logging avanzado), `ansi_term`, `matchers`, `nu-ansi-term`, `sharded-slab`, `thread_local`, `valuable`, `tracing-log`, `tracing-serde`, `tracing-subscriber` y `tracing-test` (para pruebas).

`rust-mev-engine/src/database.rs`

- Se añadió `use crate::types::KitDeArmado;`.
- Se añadió el campo `pub kit_de_armado: Option<KitDeArmado>,` a las estructuras `Opportunity` y `Execution` para almacenar los detalles del combo de arbitraje.

`rust-mev-engine/src/executor.rs`

- **Importaciones**: Se añadieron `use crate::math_engine;` y `use crate::types::{KitDeArmado, PoolReserves, DexFees, GasCostEstimator};`.
- **`execute_atomic_arbitrage`**: La lógica de ejecución se refactorizó para utilizar el `KitDeArmado`. Ahora, en lugar de construir una transacción genérica, se asegura la presencia de un `KitDeArmado` y se construye un bundle de transacciones a partir de este, enviándolo a través de un relay privado (Flashbots).
- **`build_arbitrage_transaction`**: Se modificó para incluir la simulación de transacciones (`self.simulate_transaction`) para estimar el gas de manera más precisa, añadiendo un buffer al gas estimado.
- **`send_public_transaction`**: Se añadió lógica para esperar el recibo de la transacción y verificar su estado (éxito o reversión), incrementando los contadores de monitoreo correspondientes (`increment_transactions_successful`, `increment_reverted_transactions`, `increment_transactions_failed`).
- **`send_private_transaction`**: Se refactorizó para iterar sobre los relays configurados (Flashbots, Bloxroute, MEV-Share) y enviar la transacción a través de ellos. Se añadió manejo de errores para cada relay y se incrementa `reverted_transactions` si todos los relays fallan o no hay ninguno configurado/habilitado.
- **Nuevas funciones auxiliares**: Se asume la adición de `build_kit_de_armado_bundle` y `simulate_transaction` para soportar la nueva lógica de ejecución y simulación.

`rust-mev-engine/src/mev_scanner.rs`

- **Importaciones**: Se añadieron `use crate::data_fetcher::DataFetcher;`, `use`

`crate::math_engine::MathEngine;` y `use crate::address_validator::AddressValidator;`.

- **Estructura** **`MevScanner`**: Se añadieron campos para `data_fetcher`, `math_engine` y `address_validator`.
- **`MevScanner::new`**: Se instanciaron los nuevos módulos (`DataFetcher`, `MathEngine`, `AddressValidator`).
- **`scan_dex_arbitrage`**: Se modificó para utilizar `data_fetcher.get_pool_reserves()` para obtener datos en tiempo real. La lógica de detección de oportunidades ahora invoca a `math_engine` para calcular la cantidad óptima de entrada y el beneficio, y a `address_validator` para verificar la seguridad de los contratos involucrados.
- **`detect_backrun_opportunity`** **y** **`detect_jit_opportunity`**: Se integró la validación de direcciones para asegurar que los contratos involucrados sean legítimos antes de considerar la oportunidad.

`rust-mev-engine/src/monitoring.rs`

- Se añadió un nuevo contador `reverted_transactions` para rastrear las transacciones que fallan o son revertidas.
- Se añadió el método `increment_reverted_transactions()`.
- Se inicializó el nuevo contador en `Monitoring::new()`.

`rust-mev-engine/src/types.rs`

- Se definieron las estructuras `KitDeArmado`, `PoolReserves`, `DexFees` y `GasCostEstimator` para modelar los componentes de arbitraje y los datos relevantes.

`rust-mev-engine/src/data_fetcher.rs` (Nuevo archivo)

- Implementa la lógica para interactuar con APIs como DefiLlama y Dexscreener, y con nodos RPC para obtener datos de pools, precios y otros datos de mercado en tiempo real.
- Contiene la función `get_pool_reserves` que utiliza `RpcManager` para consultar las reservas de liquidez de los contratos de DEX.

`rust-mev-engine/src/math_engine.rs` (Nuevo archivo)

- Implementa la lógica del cálculo diferencial para la optimización de oportunidades de arbitraje.
- Contiene funciones para definir la función de beneficio, calcular su derivada y encontrar el punto óptimo de entrada para maximizar las ganancias.

`rust-mev-engine/src/address_validator.rs` (Nuevo archivo)

- Implementa la lógica para validar direcciones de contratos y tokens, verificando su legitimidad y previniendo interacciones con entidades maliciosas.
- Contiene funciones como `is_safe_contract` que consultan bases de datos de seguridad o listas blancas/negras.

`rust-mev-engine/src/lib.rs`

- Se actualizaron las declaraciones de módulo para incluir `mod data_fetcher;` , `mod math_engine;` y `mod address_validator;` .

`rust-mev-engine/tests/test_mev_scanner.rs` (Nuevo archivo)

- Se creó un archivo de pruebas unitarias para `mev_scanner.rs` para verificar la correcta integración y funcionamiento de las nuevas funcionalidades, incluyendo la detección de oportunidades con el motor matemático y la validación de direcciones.

Conclusión

Las modificaciones realizadas representan una actualización integral del bot MEV, incorporando principios de cálculo diferencial para la optimización, adquisición de datos en tiempo real para la precisión, y robustas medidas de seguridad para la protección. Estos cambios están diseñados para mejorar significativamente la rentabilidad y la fiabilidad del bot en un entorno de mercado dinámico y competitivo.