Rates Module

The Maker Protocol's Rate Accumulation Mechanism * Module Name: * Rates Module * Type/Category: * Rates * Associated MCD System Diagram * Contract Sources: * *Jug * * Pot * * *

Introduction

A fundamental feature of the MCD system is to accumulate stability fees on Vault debt balances, as well as interest on Dai Savings Rate (DSR) deposits.

The mechanism used to perform these accumulation functions is subject to an important constraint: accumulation must be a constant-time operation with respect to the number of Vaults and the number of DSR deposits. Otherwise, accumulation events would be very gas-inefficient (and might even exceed block gas limits).

For both stability fees and the DSR, the solution is similar: store and update a global "cumulative rate" value (per-collateral for stability fees), which can then be multiplied by a normalized debt or deposit amount to give the total debt or deposit amount when needed.

This can be described more concretely with mathematical notation:

- Discretize time in 1-second intervals, starting fromt
- _0;
- Let the (per-second) stability fee at timet
- have valueF_i
- (this generally takes the form 1+x
- , wherex
- is small)
- Let the initial value of the cumulative rate be denoted byR
- _0
- Let a Vault be created at timet_0
- with debtD
- _0 drawn immediately; the normalized debtA
- (which the system stores on a per-Vault basis) is calculated asD
- _0/R
- _0
-

Then the cumulative rate R at time T is given by:

! R (t) ≡ R 0 ∏ i = t 0 + 1 t F i = R 0 ⋅ F t 0 + 1 ⋅ F t 0 + 2 ⋯ F t − 1 ⋅ F t R(t) \equiv R_0 \prod_{i=t_0 + 1}^{t} F_i = R_0 \cdot F_{t_0 + 1} \cdot F_{t_0 + 2} \cdots F_{t-1} \cdot F_t And the total debt of the Vault at timet would be:

! D (t) ≡ A ⋅ R (t) = D 0 ∏ t = 1 T F i D(t) \equiv A \cdot R(t) = D_0 \prod_{t=1}^{T} F_i In the actual system,R is not necessarily updated with every block, and thus actualR values within the system may not have the exact value that they should in theory. The difference in practice, however, should be minor, given a sufficiently large and active ecosystem.

Detailed explanations of the two accumulation mechanisms may be found below.

Stability Fee Accumulation

Overview

Stability fee accumulation in MCD is largely an interplay between two contracts: theVat (the system's central accounting ledger) and theJug (a specialized module for updating the cumulative rate), with theVow involved only as the address to which the accumulated fees are credited.

?

The Vat stores, for each collateral type, anIlk struct that contains the cumulative rate (rate) and the total normalized debt associated with that collateral type (Art). The Jug stores the per-second rate for each collateral type as a combination of abase value that applies to all collateral types, and aduty value per collateral. The per-second rate for a given collateral type is the sum of its particularduty and the globalbase .

CallingJug.drip(bytes32 ilk) computes an update to the ilk'srate based onduty ,base , and the time sincedrip was last called for the given ilk (rho). Then the Jug invokesVat.fold(bytes32 ilk, address vow, int rate_change) which:

- addsrate_change
- torate
- for the specified ilk
- increases theVow

- 's surplus byArt*rate_change
- increases the system's total debt (i.e. issued Dai) byArt*rate_change
- .
-

Each individual Vault (represented by anUrn struct in the Vat) stores a "normalized debt" parameter calledart . Any time it is needed by the code, the Vault's total debt, including stability fees, can be calculated asart*rate (whererate corresponds to that of the appropriate collateral type). Thus an update toIlk.rate viaJug.drip(bytes32 ilk) effectively updates the debt for all Vaults collateralized withilk tokens.

Example With Visualizations

Suppose at time 0, a Vault is opened and 20 Dai is drawn from it. Assume thatrate is 1; this implies that the storedart in the Vault'sUrn is also 20. Let thebase andduty be set such that after 12 years,art*rate = 30 (this corresponds to an annual stability of roughly 3.4366%). Equivalently,rate = 1.5 after 12 years. Assuming thatbase + duty does not change, the growth of the effective debt can be graphed as follows:

?

Now suppose that at 12 years, an additional 10 Dai is drawn. The debt vs time graph would change to look like:

?

Whatart would be stored in the Vat to reflect this change? (hint:not 30!) Recall thatart is defined from the requirement thatart * rate = Vault debt. Since the Vault's debt is known to be 40 andrate is known to be 1.5, we can solve forart : 40/1.5 ~ 26.67.

Theart can be thought of as "debt at time 0", or "the amount of Dai that if drawn at time zero would result in the present total debt". The graph below demonstrates this visually; the length of the green bar extending upwards from t = 0 is the post-drawart value.

?

Some consequences of the mechanism that are good to keep in mind:

- There is no stored history of draws or wipes of Vault debt
- There is no stored history of stability fee changes, only the cumulative effectiverate
- Therate
- value for each collateral perpetually increases (unless the fee becomes negative at some point)
-

Who callsdrip ?

The system relies on market participants to calldrip rather than, say, automatically calling it upon Vault manipulations. The following entities are motivated to calldrip :

- Keepers seeking to liquidate Vaults (since the accumulation of stability fees can push a Vault's collateralization ratio into unsafe territory, allowing Keepers to liquidate it and profit in the resulting collateral auction)
- Vault owners wishing to draw Dai (if they don't calldrip
- prior to drawing from their Vault, they will be charged fees on the drawn Dai going back to the last timedrip
- was called—unless no one callsdrip
- before they repay their Vault, see below)
- MKR holders (they have a vested interest in seeing the system work well, and the collection of surplus in particular is critical to the ebb and flow of MKR in existence)
-

Despite the variety of incentivized actors, calls todrip are likely to be intermittent due to gas costs and tragedy of the commons until a certain scale can be achieved. Thus the value of therate parameter for a given collateral type may display the following time behavior:

?

Debt drawn and wiped betweenrate updates (i.e. betweendrip calls) would have no stability fees assessed on it. Also, depending on the timing of updates to the stability fee, there may be small discrepancies between the actual value ofrate and its ideal value (the value ifdrip were called in every block). To demonstrate this, consider the following:

- at t = 0, assume the following values:
-

! rate = 1 ; total fee = f \text{rate} = 1 \text{ ; total fee} = f in a block with t = 28,drip is called—now:

! rate = f 28 \text{rate} = f^{28} in a block with t = 56, the fee is updated to a new, different value:

! totalfee → g \text{totalfee} \xrightarrow{} g in a block with t = 70,drip is called again; the actual value ofrate that obtains is:

! rate = f 28 g 42 \text{rate} = f^{28} g^{42} however, the "ideal"rate (ifdrip were called at the start of every block) would be:

! rate i d e a l = f 56 g 14 \text{rate}_{ideal} = f^{56}g^{14} Depending on whetherf

> g org f , the net value of fees accrued will be either too small or too large. It is assumed thatdrip calls will be frequent enough such inaccuracies will be minor, at least after an initial growth period. Governance can mitigate this behavior by callingdrip immediately prior to fee changes. The code in fact enforces thatdrip must be called prior to aduty update, but does not enforce a similar restriction forbase (due to the inefficiency of iterating over all collateral types).

Dai Savings Rate Accumulation

Overview

DSR accumulation is very similar to stability fee accumulation. It is implemented via thePot , which interacts with the Vat (and again the Vow's address is used for accounting for the Dai created). The Pot tracks normalized deposits on a per-user basis (pie[usr]) and maintains a cumulative interest rate parameter (chi). Adrip function analogous to that of Jug is called intermittently by economic actors to trigger savings accumulation.

?

The per-second (or "instantaneous") savings rate is stored in thedsr parameter (analogous tobase+duty in the stability fee case). Thechi parameter as a function of time is thus (in the ideal case ofdrip being called every block) given by:

! chi (t) ≡ chi 0 ∏ i = t 0 + 1 t dsr i \text{chi}(t) \equiv \text{chi}0 \prod{i=t_0 + 1}^{t} \text{dsr}_i where chi_0 is simply chi(t _0).

Suppose a user joinsN Dai into the Pot at timet _0. Then, their internal savings Dai balance is set to:

! pie[usr] = N / chi 0 \text{pie[usr]} = N / \text{chi}_0 The total Dai the user can withdraw from the Pot at timet is:

! pie[usr] · chi (t) = N ∏ i = t 0 + 1 t dsr i \text{pie[usr]} \cdot \text{chi}(t) = N \prod_{i=t_0 + 1}^{t} \text{dsr}_i Thus we see that updates tochi effectively increase all Pot balances at once, without having to iterate over all of them.

After updatingchi ,Pot.drip then callsVat.suck with arguments such that the additional Dai created from this savings accumulation is credited to the Pot contract while the Vow'ssin (unbacked debt) is increased by the same amount (the global debt and unbacked debt tallies are increased as well). To accomplish this efficiently, the Pot keeps track of a the total sum of all individualpie[usr] values in a variable calledPie .

Notable Properties

The following points are useful to keep in mind when reasoning about savings accumulation (all have analogs in the fee accumulation mechanism):

- ifdrip
- is called only infrequently, the instantaneously value ofchi
- may differ from the ideal
- the code requires thatdrip
- be called prior todsr
- changes, which eliminates deviations ofchi
- from its ideal value due to such changes not coinciding withdrip
- calls
- chi
- is a monotonically increasing value unless the effective savings rate becomes negative (dsr
- <ONE
-)
- There is no stored record of depositing or withdrawing Dai from the Pot
- There is no stored record of changes to thedsr
-

Who callsdrip ?

The following economic actors are incentivized (or forced) to callPot.drip :

- any user withdrawing Dai from the Pot (otherwise they lose money!)
- any user putting Dai into the Pot—this is not economically rational, but is instead forced by smart contract logic that requiresdrip
- to be called in the same block as new Dai is added to the Pot (otherwise, an economic exploit that drains system surplus is possible)

- any actor with a motive to increase the system debt, for example a Keeper hoping to trigger flop (debt) auctions
-

A Note On Setting Rates

Let's see how to set a rate value in practice. Suppose it is desired to set the DSR to 0.5% annually. Assume the real rate will track the ideal rate. Then, we need a per-second rate valuer such that (denoting the number of seconds in a year byN):

! r N = 1.005 r^N = 1.005 An arbitrary precision calculator can be used to take theN -th root of the right-hand side (withN = 31536000 = 36524 60*60), to obtain:

! r = 1.000000000158153903837946258002097... r = 1.000000000158153903837946258002097... Thedsr parameter in the Pot implementation is interpreted as aray , i.e. a 27 decimal digit fixed-point number. Thus we multiply by 10^27 and drop anything after the decimal point:

! dsr = 1000000000158153903837946258 \text{dsr} = 1000000000158153903837946258 Thedsr could then be set to 0.5% annually by calling:

Pot.file("dsr", 1000000000158153903837946258)

Previous Chief - Detailed Documentation Next Pot - Detailed Documentation Last updated4 years ago On this page * Introduction * Stability Fee Accumulation * Overview * Example With Visualizations * Who calls drip? * Dai Savings Rate Accumulation * Overview * Notable Properties * Who calls drip? * A Note On Setting Rates

Export as PDF